A Field Programmable Sequencer and Memory with Middle Grained Programmability Optimized for MCU Peripherals

INPUT THE TYPE OF MANUSCRIPT

A Field Programmable Sequencer and Memory with Middle Grained Programmability Optimized for MCU Peripherals

Yoshifumi KAWAMURA[†,††a)], *member*, Naoya OKADA[†,†††], *member*, Yoshio MATSUDA[†], *non-member*, Tetsuya MATSUMURA[4], *member*, Hiroshi MAKINO[5], *member*, and Kazutami ARIMOTO[6], *member*

SUMMARY A Field Programmable Sequencer and Memory (FPSM), which is a programmable unit exclusively optimized for peripherals on a micro controller unit, is proposed. The FPSM functions as not only the peripherals but also the standard built-in memory. The FPSM provides easier programmability with a smaller area overhead, especially when compared with the FPGA. The FPSM is implemented on the FPGA and the programmability and performance for basic peripherals such as the 8 bit counter and 8 bit accuracy Pulse Width Modulation are emulated on the FPGA. Furthermore, the FPSM core with a 4K bit SRAM is fabricated in 0.18 μm 5 metal CMOS process technology. The FPSM is an half the area of FPGA, its power consumption is less than one-fifth.

key words: *FPGA, MCU peripherals, field programmable devices, sequencer, SRAM.*

1. Introduction

Nowadays Micro Controller Units (MCUs), or microcomputers, are widely used in a variety of different systems and for various applications. The various needs of many customers and systems require many types of chips and MCU vendors provide MCU chips as families for these requirements. Furthermore, the individual needs of customers and system requirements generates various chips even in a family, called products lineup, with different memory capacities, memory organizations, and/or peripherals although the chips have the same basic architecture. Amongst them, requirements for peripherals are extensive. Usually, standard peripherals are provided as hard wired logics on the MCU. However, some customers will require more than one timer or an FIFO memory in addition to the peripherals provided. Other customers might need multiple serial interfaces. These customers' requirements increase products lineup.

Recently, the MCU markets for sensor network systems and medical applications have been growing rapidly. These systems will require even a greater variety of MCU peripherals [1]-[6], so realizing these variations on one MCU chip is very important. One solution is to embed programmable devices for MCU peripherals. If MCU peripherals can be easily configured on programmable devices, it will drastically reduce the number of products lineup and greatly improve productivity, cost, and quick turn around time (QTAT). Thus, programmable peripherals will be important for future MCUs.

A straightforward solution is to embed a small Field Programmable Gate Array (FPGA) like core connected to an internal peripheral bus on the MCU. The FPGA is a fine grained programmable device, however, great overheads exist in its areas, cost and performance in compensation for the fine grained programmability when compared to ASIC in the same function under the same process technology. In addition, MCU vendors must develop their original FPGA like core and dedicated mapping tools. A large number of metal layers is also necessary for this small core, increasing a cost. On the other hand, re-configurable coarse grained processors are proposed. Reconfigurable processors are programmable devices mainly at the application level [7]-[10], therefore, allow the flexible execution of various kinds of applications. However, the coverage area of re-configurable processors has remained limited. This programmable architecture is not applicable to our purpose which realizes various peripherals on one MCU chip. One other approach is a look–up table cascade architecture composed of a serial connection of large scale memories [11].

This paper proposes a Field Programmable Sequencer and Memory (FPSM), which is an MCU with programmable parts for MCU peripherals. Since peripheral functions can be easily configured by users in the field after shipping [12], the FPSM can be said to provide the "user structured" MCUs. Peripherals are programmed through an array of functional memory units, consisting of an SRAM and a small address control unit. After analyzing the customer's requirements, the address control unit is optimized for the MCU peripherals, realizing middle range

†The authors are with the College of Science and Engineering, Kanazawa University, Kanazawa-shi, 920-1192, Japan.
††The author is also with SKY Technology Company Limited, Saitama-shi, 337-0008, Japan.
†††The author is now with the Information Technology R & D Center, Mitsubishi Electric Corporation, Kamakura-shi, 247-8501, Japan.
4The author is with the College of Engineering, Nihon University, Koriyama-shi, 963-8642, Japan.
5The author is with the Faculty of Information Science and Technology, Osaka Institute of Technology, Hirakata-shi, 573-0196, Japan.
6The author is with he Faculty of Computer Science and Systems Engineering, Okayama Prefectural University, Soja-shi, 719-1197, Japan.
a)E-mail: yoshifumi.kawamura.g@gmail.com.

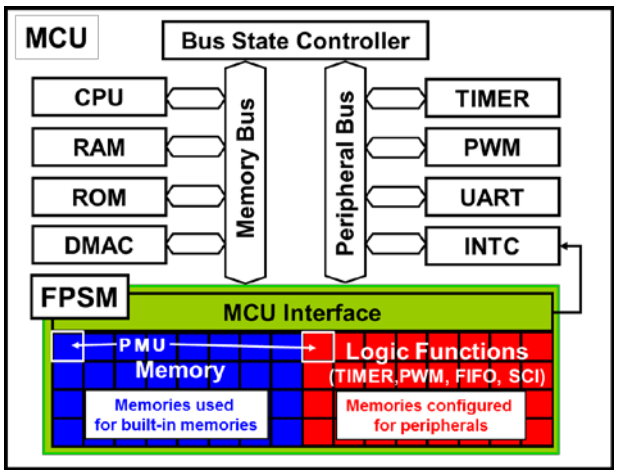


Fig. 1 Block diagram of the Micro Controller Unit (MCU) with the FPSM.

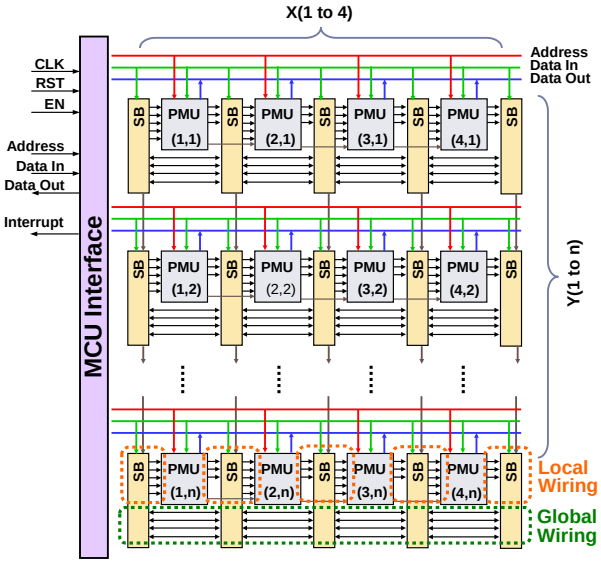


Fig. 2 The FPSM is the two dimensional array of the Programmable Memory Units (PMU) and Switch Boxes (SB).

grained programmability and precise control. Functional memory units can also be used as the standard built-in memory if the units are not configured for peripherals. The FPSM has the dual functions of programmable peripherals and standard memory. Thus, the FPSM can provide the highest gate density and the easiest configuration. Since the configuration is performed through standard memory write operations, dedicated mapping tools are unnecessary.

The rest of the paper is organized as follows. In Sec. 2, the concept and architecture of the FPSM are described. In Sec. 3, the functional blocks of the FPSM are explained. In Sec. 4, the configuration and operations of the FPSM are described. The implementation of the FPSM on the MCU is explained in Sec. 5. The configuration flow and operations of the FPSM are also given in Sec. 5. The FPSM is implemented on the FPGA and major MCU peripherals such as the Counter, Timer, Pulse Width Modulation (PWM), and First In First Out (FIFO) memory are emulated on the FPSM in the FPGA. The implementation and emulation results are described in Sec. 6. The LSI implementation of the FPSM and the measurement results of the test chip are described in Sec. 7. A comparison of the FPSM and the FPGA is given as well. Finally, the conclusion is given in Sec. 8.

2. Concept and Architecture of the FPSM

The major components of the MCU are the Central Processing Unit (CPU), the memory including RAM, ROM and non-volatile RAM, and peripherals such as the Timer, Pulse Width Modulation (PWM), and Universal Asynchronous Receiver Transmitter (UART). The proposed Field Programmable Sequencer and Memory (FPSM) is a programmable device optimized exclusively for MCU peripherals.

The FPSM is embedded on the MCU. A conceptual block diagram of an MCU with the FPSM is shown in Fig. 1. The FPSM is connected to the Memory Bus or the Peripheral Bus through the MCU Interface. Data are written to the memory through the MCU Interface by the CPU. The CPU writes the configuration data for the desired peripheral to the memory in the FPSM only once when booting up the MCU, and after that, the FPSM operates autonomously with the clock signal CLK. The FPSM usually does not access the CPU, so the operations of the configured peripherals add no load to the CPU.

The FPSM consists of a programmable array, called a Programmable Memory Unit (PMU), as shown in Fig. 1. The detailed structure is given in Fig. 2. The FPSM comprises MCU Interface, multiple PMUs and Switch Boxes (SB). The MCU Interface includes the PMU array decoder, which designates a coordinate of the PMU by a global address. This is a part of the address sent from the CPU. Multiple PMUs can be connected in parallel and/or cascade connections through SBs. The SB connects neighboring PMUs by local wires and multiple PMUs by global wires. PMU connections are controlled by information stored at the registers in the SBs.

When a PMU operates as a peripheral, data are transmitted to or received from the CPU through the Peripheral Bus. On the other hand, unconfigured memory for the peripherals can be accessed through the Memory Bus. Even if some of the PMUs are operating as peripherals, the remaining PMUs can operate at the same time as the standard built-in memory. That is, the PMU functions as both the memory and the peripherals.

3. Functional Blocks of the FPSM

In this section, we describe the functional blocks of the PMUs; the MCU Interface, PMU, and SB.

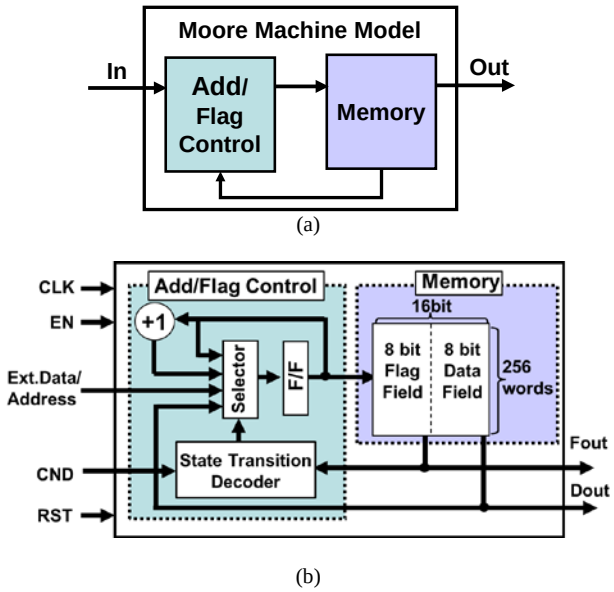


Fig. 3 (a) The Moore Machine of sequential circuits and a (b) block diagram of the PMU in the FPSM.

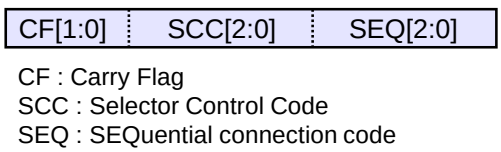


Fig. 4 Flag Field format.

3.1 MCU Interface

The main role of the MCU Interface is to maintain signal and timing compatibility between PMUs and an equipped MCU. The MCU Interface translates an address from the MCU to a global address and a local address and distributes them to PMUs through the address bus as shown in Fig. 2. The global address selects one PMU and the local address is taken into the selected PMU as an external address.

Cycle timing adjustment and synchronization between the signals of the MCU and PMUs are also done at this interface depending on the MCU. For example, the number of write cycles and read cycles for the MCU are controlled to meet memory (SRAM) cycles in the PMU. The number of the cycles is adjusted at a register in the MCU Interface. In addition, signals from the MCU are translated to signals for the PMU and vice versa. For example, one Data Out bus is selected from multiple Data Out buses for the PMU array and the data are sent to the MCU after cycle adjustment.

3.2 Programmable Memory Unit (PMU)

The PMU contains a memory unit (Memory) and a small logic unit (Add/Flag Control) that controls the memory address. Various sequencers and combinational logics can

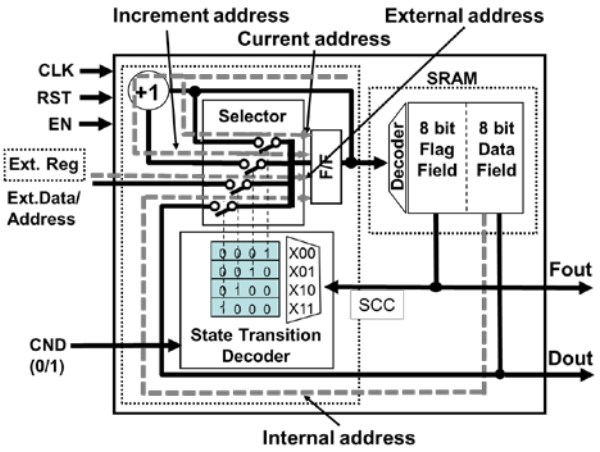


Fig. 5 Four kinds of address paths.

be configured by storing truth table data in the Memory. Fig. 3(a) shows a standard sequential circuit model, the Moore-Machine model [13]-[15], including the feedback loop from the memory to the Add/Flag Control unit. The basic concept of the PMU is the Moore-Machine. The PMU is a clock synchronized memory with an autonomous address control, which also operates as a state machine [14], [15].

Fig. 3(b) shows a block diagram of the PMU. One PMU has a 4 Kbit SRAM of 256 words by 16 bits. A word of 16 bits is segmented into two fields, the 8 bit Flag Field and the 8 bit Data Field. The Flag Field and the Data Field correspond to the operation code and the operand of a microcode, respectively. Thus, the Flag Field stores information for controlling the Selector through the State Transition Decoder. The bit assignment of the Flag Field is shown in Fig. 4. The first two bits CF[1:0] are assigned to the Carry Flag (CF). The second three bits SCC[2:0] are the Selector Control Code and are mainly used for controlling the Selector. The last three bits SEQ[2:0] are used to control the other PMUs in the case of multiple PMU connections. Addresses of the next state are stored in the Data Field when the PMU is used as a sequencer. When the PMU is used as a combinational logic circuit, the truth table data are stored in this field. The details will be explained later using examples.

The condition signal (CND), external data (Ext. Data), and external addresses (Ext. Address) are inputted to the PMU. The Selector chooses one of the four addresses; the external address, the internal address read out from the Data Field, the current address, or the incremented address of the current address, depending on the CND and the SCC bits read out from the Flag Field. The selector control scheme of the State Transition Decoder and the address paths are shown in Fig. 5. The address paths are controlled by the SCC bits inputted to the State Transition Decoder. The State Transition Decoder chooses a switch at the selector and controls the on/off of the switch using two out of three SCC bits. The one remaining bit is reserved for

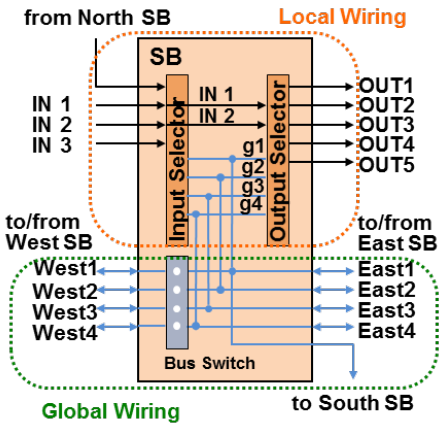


Fig. 6 Block diagram of Switch Box (SB) on FPSM.

future extension. The four kinds of address transitions are a result of analyzing the MCU peripherals and cover almost every peripheral frequently required in the field. This limitation simplifies the Address/Flag Control unit and keeps its area as small as possible.

The selected address is sent to the memory, and the data stored in the Flag Field and Data Field are read out from the memory. The read out data (Fout) from the Flag Field are sent to the State Transition Decoder and used for selecting the next address. The read out data (Dout) from the Data Field are used as the internal address in the next cycle. Some or all of the Dout are also sent to the subsequent PMU and are used for the address of the PMU when there are multiple connections.

3.3 Switch Box (SB)

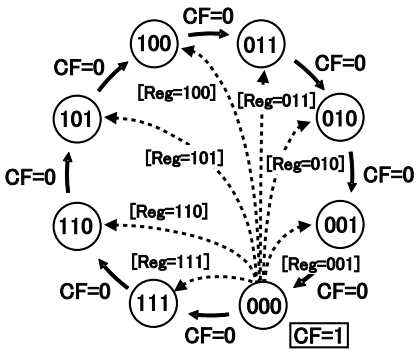
Fig. 6 shows a block diagram of the SB. Signals between PMUs are exchanged through four 4-bit width global wires. Signal connection to the PMUs is controlled by the Bus Switch for global wires and by the Input and Output selector for the local wires. The local wiring is unidirectional, while the global wiring is bi-directional. PMU input and output signals are also treated in units of 4 bits. Three signals (IN1, IN2, and IN3) having 4 bit width from the left PMU and one signal having 4 bit width from the upper SB (north SB) are input to the SB. These signals are selected at the Input Selector and outputted to the global wires and to the Output Selector. One global wire is connected to the lower SB (south SB). The two signals (IN1 and IN2) are directly input to the Output Selector for the appropriate PMU in a cascade connection. The Output Selector sends five signals to the right PMU by choosing five signals among the two signals from the Input Selector and the four signals on the global wires.

4. Configuration

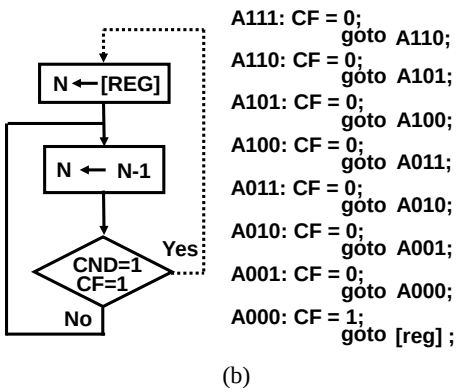
In this section, the configuration of the FPSM is explained

Address selection	Address Address[2:0]	Flag Field CF[1]	Data Field Data[2:0]
Internal Add	000	1	111
Internal Add	001	0	000
Internal Add	010	0	001
Internal Add	011	0	010
Internal Add	100	0	011
Internal Add	101	0	100
Internal Add	110	0	101
Ext. Add [Reg]	111	0	110

Fig. 7 Configuration data of the 3 bit down counter and its address selections.



(a)



(b)

Fig. 8 (a) State transitions on the N-ary counter and (b) operation flow of the N-ary counter.

by using examples. The first example is the 3 bit down counter. Fig. 7 shows the configuration data in the Flag Field and Data Field. Irrelevant bits are omitted. The first column is the address chosen out of the four address paths at the Selector. The second column is the address of the Memory in the PMU and the third column is the first bit CF[1] out of CF[1:0]. CF=1 for only the address “000”. Finally, the fourth column is the value of the Data Field, which is the decrement value of the memory address.

When the signal EN is asserted by the CPU as the trigger, the count operation automatically starts with the

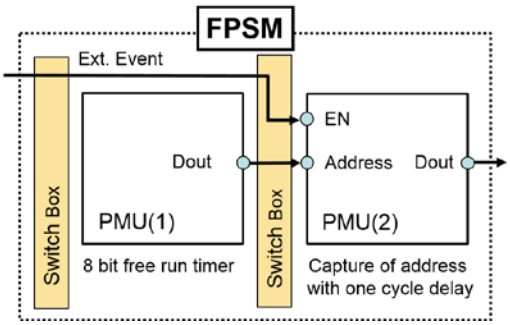


Fig. 9 Capture of a single event using the PMU.

PMU (1)

Address	Data[7:0]
.....	.....
1000 0110	1000 0111
1000 0111	1000 1000
1000 1000	1000 1001
1000 1001	1000 1010
.....	.....

Dout

PMU (2)

Address	Data[7:0]
.....	.....
1000 0110	1000 0110
1000 0111	1000 0111
1000 1000	1000 1000
1000 1001	1000 1001
.....	.....

Dout

Fig. 10 Configuration data of the PMUs for the single event capture function shown in Fig. 9.

clock signal CLK. For example, if “111” is inputted to the PMU from the external register, the value “111” is taken in the Memory of the PMU as the external address and the data stored in the Flag Field and Data Field are read out from the Memory. In the next cycle, the read out data “110” from the Data Field are used as the internal address of the Memory. The State Transition Decoder controls the Selector to adopt these data as the internal address, referring to CF[1] and the SCC bits. Since CF=0 except for the address “000”, the same operation is repeated until the address reaches “000”. At the address “000”, the count operation stops because CF=1. If necessary, the CPU can obtain the current counted value by reading the Dout. In our design, a counter of up to 32 bits can be constructed using four cascaded PMUs.

If N ($N \leq 8 = 2^3$) is set at the external register, the N-ary counter is realized. The state transitions and operation flow are shown in Figs. 8(a) and (b), respectively. For example, if “101” is inputted from the external register, the PMU counts down automatically and stops in six cycles, giving CF=1 in the Flag Field and the address “000” in the Data Field. The MCU Interface generates the interrupt signal INT, referring to CF=1 and notices the completion of the count operation to the CPU with the interruption signal INT.

The capture function, which gets the cycle when an event occurs, is realized using two connected PMUs, as shown in Fig. 9. The configurations are shown in Fig. 10. The 8 bit up counter is configured on PMU (1). On PMU (2), the address itself is stored at an address of PMU (2). The signal Ext. Event represents an event and becomes “1” when the event occurs. This signal is connected to the

Table 1 The PMU resources for the MCU peripheral examples.

Main Functions	Variations	Number of PMUs				
		1	2	3	4	8
Memory (Random/Serial Access)	8 bits	✔				
	16 bits		✔			
	24 bits			✔		
	32 bits				✔	
Counter/Timer	8 bits (Up/Down)	✔				
	16 bits (Up/Down)		✔			
	24 bits (Up/Down)			✔		
	32 bits (Up/Down)				✔	
Shifter	8 bits (Logic/Arithmetic)	✔				
	8 bits (Left/Right Rotate)	✔				
	16 bits (Logic/Arithmetic)		✔			
	16 bits (Left/Right Rotate)		✔			
	24 bits (Logic/Arithmetic)			✔		
	24 bits (Left/Right Rotate)			✔		
	32 bits (Logic/Arithmetic)				✔	
	32 bits (Left/Right Rotate)				✔	
Clocked Serial Interface	Receiver			✔		
	Transmitter				✔	
FIFO Memory (R/W Buffer size)	Up to 16 Bytes		✔			
	Up to 256 Bytes			✔		
Calculation Unit (Add:Adder, Sub:Subtracter)	4 bits Add/Sub	✔				
	8 bits Add/Sub		✔			
	12 bits Add/Sub			✔		
	16 bits Add/Sub				✔	
PWM	8 bit resolution (1 Output)			✔		
	16 bit resolution (2 Output)					✔

Note; Check marks represent the required number of PMUs.

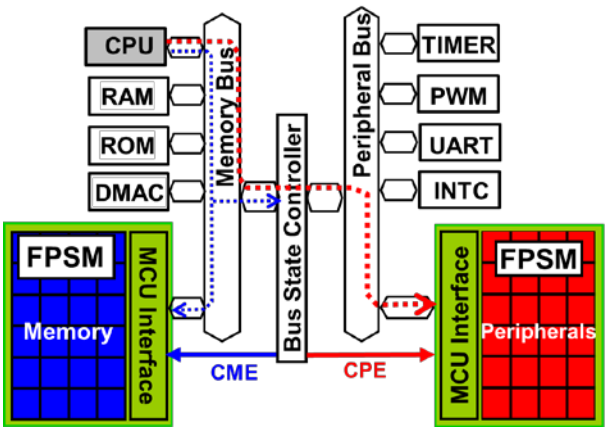


Fig. 11 Bus architecture of the FPSM; The connection to the internal Memory Bus for the unconfigured FPSM and the connection to the Peripheral Bus for the configured FPSM.

enable signal EN port of PMU (2). The Dout of PMU (1) is connected to the address port of PMU (2). This address is selected as the external address for PMU (2). When the Ext. Event becomes “1”, the address of PMU (1) at that time is taken in PMU (2) because the EN of PMU (2) becomes “1”. This address is recognized after a one cycle delay by receiving the Dout of PMU (2).

Combinational logic circuits are also available by configuring the truth table data on the PMU. Various functions can be configured by combining the configuration data and the multiple parallel and/or cascade connections of the PMU. Users can configure the calculation unit if they wish to. Examples of configurable basic functions are summarized in Table 1. Thus, the FPSM

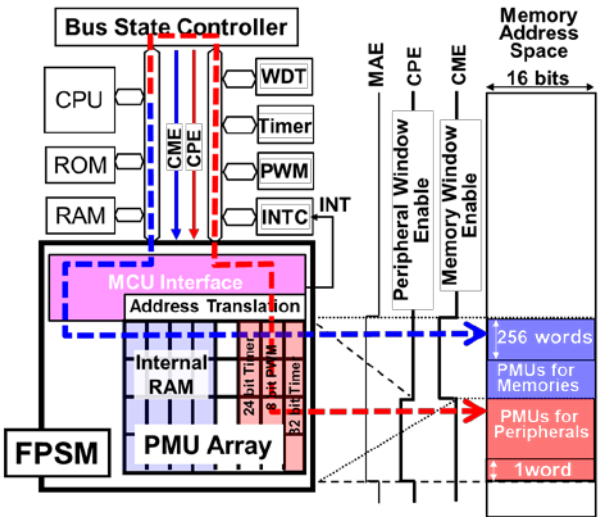


Fig. 12 Accesses to FPSMs in the standard built-in memory and peripherals.

covers almost all MCU peripherals.

5. Implementation of the FPSM on the MCU

In this Section, the implementation and configuration flow of the FPSM on an MCU are described. As stated in Sec. 2, the FPSM is connected to the Memory Bus or the Peripheral Bus through the MCU Interface. The bus connections are shown in Fig. 11 (see also Fig. 1). The PMUs that are not configured for peripherals are used as the standard build-in memory. The PMUs are connected to the internal Memory Bus through the MCU Interface in the FPSM. If the PMUs are configured as peripherals, they are connected to the Peripheral Bus in the MCU through the MCU Interface. The Peripheral Bus is connected to the internal Memory Bus by the Bus State Controller, a kind of Bridge. The connection is managed by the Bus State Controller. The connections of the FPSM to the Memory and Peripheral Buses are activated by the signal CME and CPE, respectively.

The FPSM is managed in the common address space of the MCU. The MCU Interface manages the addresses of the memory and peripherals on the FPSM in the I/O mapped method. This memory address mapping is shown in Fig. 12. The FPSM is mapped on a part of the MCU address space. The address space of the FPSM is controlled by the Memory Access Enable (MAE) signal. Furthermore, the address space of the FPSM is managed by the Memory Window Enable signal (CME) and the Peripheral Window Enable signal (CPE), whether the PMUs are configured or not. An unconfigured PMU is mapped as the 256 word standard memory because the PMU has 256 words. If the PMU is configured as a peripheral, the PMU is mapped on the address space as a one word memory, of which address is the first address of the 256 words, because the PMU

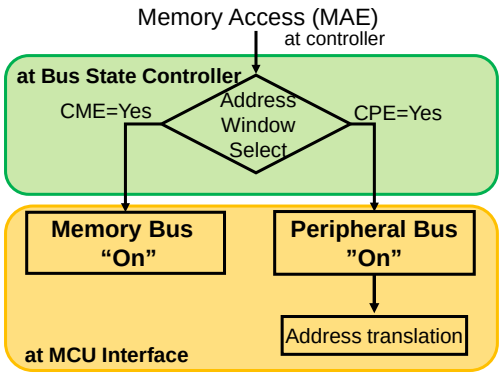


Fig. 13 Memory Bus and Peripheral Bus access control scheme of the FPSM at the MCU Interface.

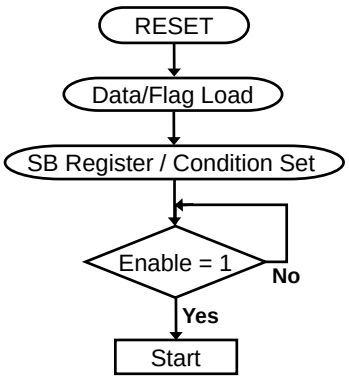


Fig. 14 Configuration flow of the FPSM on the MCU.

functions as one peripheral. The mapped address of the PMU is equivalent to the register address that specifies the standard peripheral in the memory mapped I/O method.

When the CPU accesses the memory, the controller (not shown in Fig. 12) generates the signal MAE if the virtual address from the CPU is in the address space of the FPSM. After receiving the MAE and the address, the Bus State Controller generates the CME or the CPE according to the address in the memory address space or in the peripheral address space, as shown in Fig. 13. The MCU Interface connects the PMU to the Memory Bus with the CME or to the Peripheral Bus with the CPE. Finally, the address is translated to the physical address of the PMU, thus keeping the compatibility between the various MCUs and the FPSM. This means that the MCU Interface circuit has to be slightly modified for every MCU, because the MCU Interface is depend on the MCU.

The FPSM is configured on an MCU in the following sequence. The configuration flow is shown in Fig. 14.

1) The PMU to be configured and its associated registers in the SBs are reset. The PMUs operate as the standard built-in memory in the default at the initial state.
2) Next, the configuration data, the Flag Field data, and the Data Field data are written into the memory in the PMU memory using standard memory write operations.

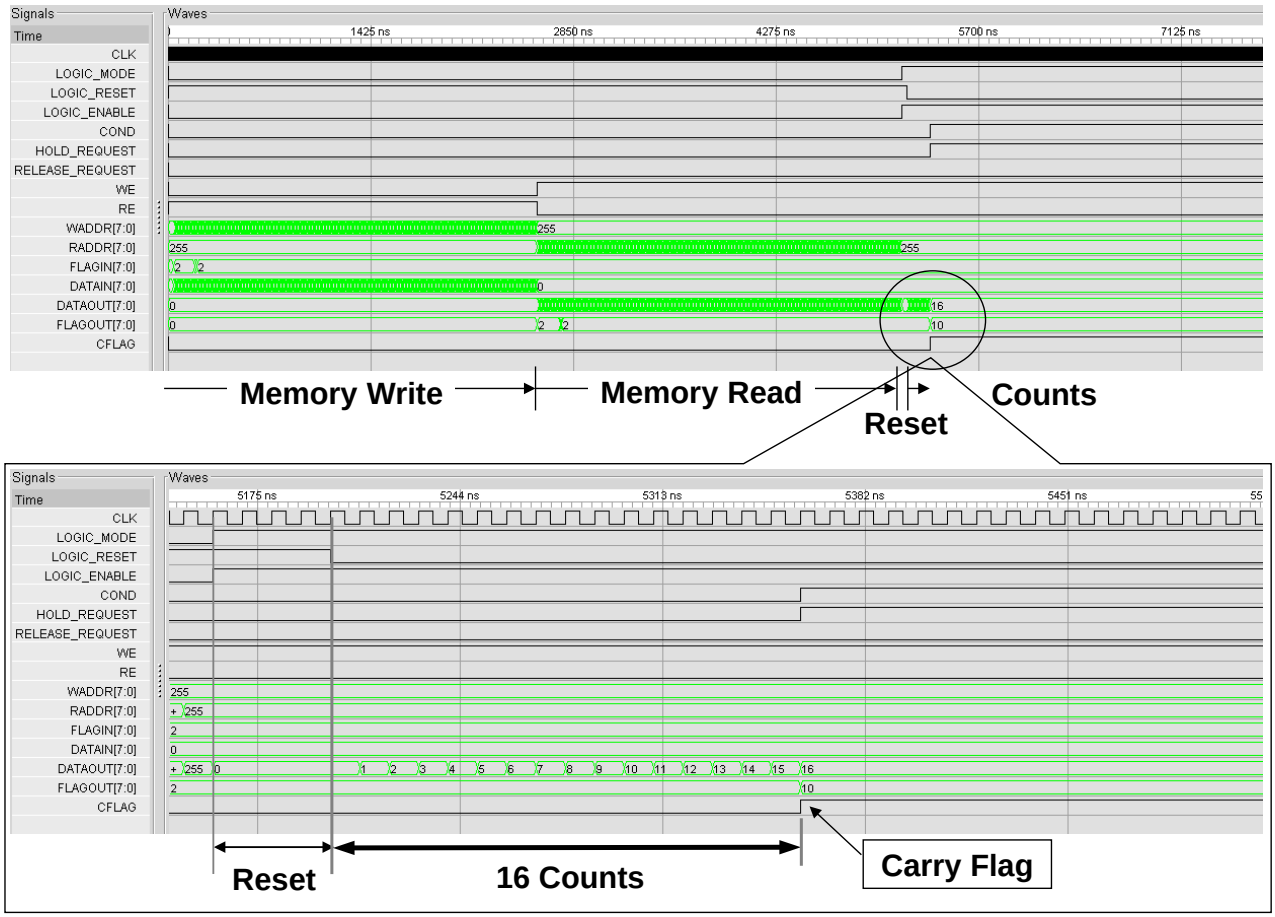


Fig. 15 Simulation results of the onetime 16 count operation on the 8 bit up counter.

3) The routing data and the mode selection (memory or peripheral) data are loaded onto the associated registers.

After the configuration, the configured PMUs operate autonomously with the clock CLK and the external event signal (Ext. Event). The configuration data and register data are stored in a built-in non-volatile memory such as the flash memory, and loaded to the PMU and the registers, respectively when the MCU is booted up.

6. Emulation of the FPSM on the FPGA

The FPSM is modeled and simulated using SystemC language. The simulation results of the 16-ary counter are shown in Fig. 15. During the first 256 memory write cycles, the PMU is configured. In order to verify that the configuration data is written correctly, the 256 memory read cycles are used (this step is not necessary for normal operations). After the reset cycles, the Carry Flag is outputted in 16 cycles. The onetime 16 count operation is performed correctly. The reset cycles are also not necessary for normal operations. The FPSM can operates immediately after the configuration.

As a more complex example, we show the simulation results of the 8 bit accuracy PWM function. The 8 bits accuracy PWM is configured on three cascaded PMUs, as shown in Fig. 16. The configurations of these three PMUs are essentially the 8 bit down counter. The first PMU is the Divider, which determines the resolution. The second controls the period and the third controls the pulse width.

The frequency f_{CLK} of the system clock (CLK) given to the first PMU is divided to C times of the CLK period corresponding to the pre-set value of C at the C register. The resolution a is

$$a = (1/f_{CLK})*C.$$

This value determines the pulse of CFLAG1. Since the value “5” is set at the C register, CFLAG1 appears after every 5 cycles of the CLK. The second PMU determines the period of a total pulse in the unit of resolution a. The pulse period t is given as

$$t= a*T.$$

The value T at the T register determines the period of the total pulse width. In Fig. 16, CFLAG 2, the period of the total pulse, appears after every 10 cycles of CFLAG1, because the value of T is “10”. Finally, the pulse width w is determined by the third PMU.

The pulse width w is

$$w=a*X,$$

where X at the X register is related to the pulse width ratio. X determines CFLAG3 which then defines the “low period”

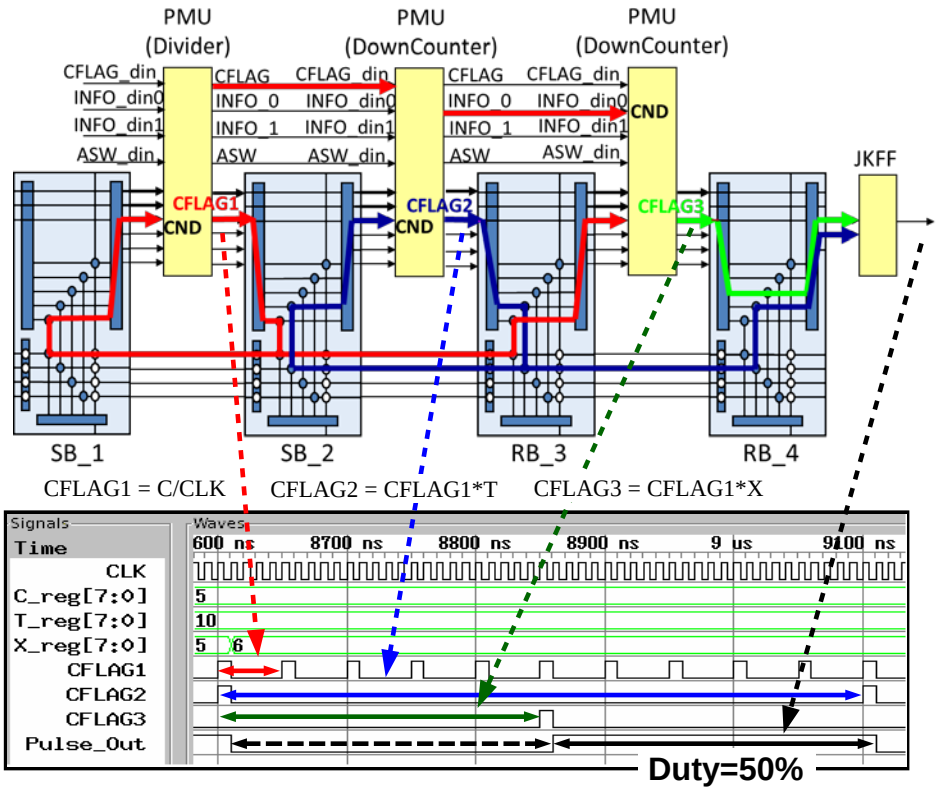


Fig. 16 Simulation results of the 8 bit PWM using three cascaded PMUs.

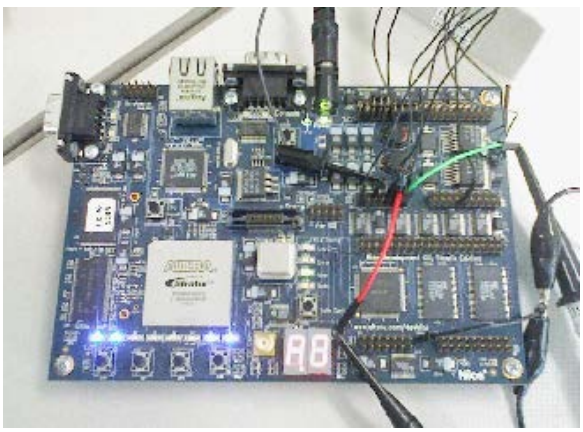


Fig. 17 Photograph of the FPGA board used for emulating the FPSM.

of the pulse. The ratio of the low period is given by X/T. CFLAG3 appears after every 5 cycles of CFLAG1 because of the value of "5" at the X register. Finally, a PWM pulse with the duty ratio of 50 % is generated, as shown in Fig. 16.

The FPSM is also implemented on the FPGA and emulated. The FPSM is coded in the Verilog HDL and mapped on the FPGA. A photograph of the FPGA board with an ALTERA Stratix II is shown in Fig. 17. The 8 bit accuracy PWM is emulated on the three cascaded PMUs mapped on the FPGA. The mapped circuit is not a hard

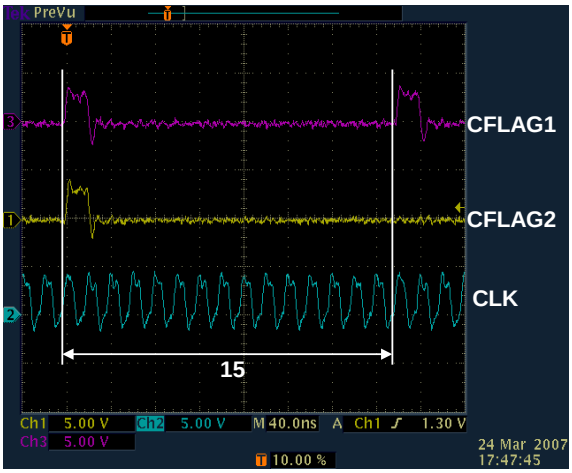
Table 2 Mapping results of the FPSM on FPGA.

FPGA		ALTERA EP1S40F780C5
Logic Synthesis		ALTERA Quartus II 6.1
System Clock		50 MHz
PWM Func.	8 bit Accuracy	300 ns @ C=15
	256 Step/Cycle	3 μs @ T=10
	Number of LEs	345
	Memory	4 K bits x 3

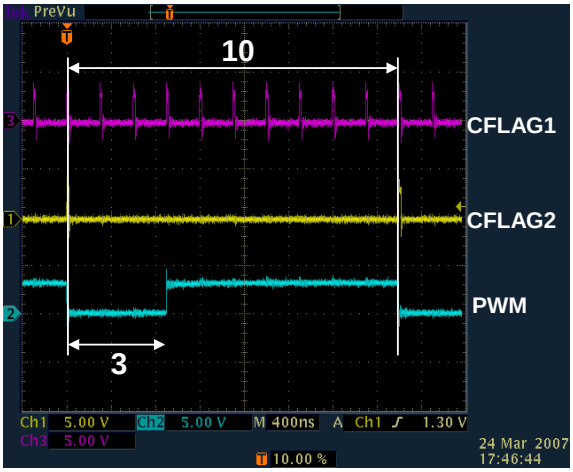
wired 8 bit accuracy PWM but three connected PMUs directly connected to each other without SBs. The mapping results are shown in Table 2. The 345 LEs (Logic Element) and three 4 Kbit SRAM are used.

In addition to verifying the architecture design, the purpose of mapping the FPSM on the FPGA is also to estimate electrical performance to be implemented on an LSI. The global wires limit the operation speed of the FPSM. Although the lengths of the global wires can be controlled using a P&R tool in an LSI implementation, it is difficult to control the global wires on the FPGA. If the global wires are also mapped onto the FPGA through the SBs, we cannot obtain useful information about operation speed. Therefore, we directly connected PMUs in the FPGA implementation.

Fig. 18 shows the observed PWM waveforms. The



(a)



(b)

Fig. 18 Observed waveforms of the 8 bit accuracy PWM emulated on the FPGA board; (a) the CFLAG1 and CFLAG2 and (b) the generated PWM pulse.

register values are C=15, T=10, and X=3. When the trigger signal is inputted, the PWM starts and the pulse width control operates, as shown in Fig. 16. The pulse of CFLAG1 appears after every 15 cycles of the CLK and CFLAG2 after every 10 cycles of CFLAG1. Finally, a pulse PWM with a duty ratio of 30 % (=X/T) is generated according to the register values “C”, “T”, and “X”. The expected waveforms are observed.

7. Test Chip

To verify our proposed architecture, an experimental FPSM core, meaning one PMU, was designed and fabricated in 0.18 µm process technology with 5 metal layers. A photomicrograph of the core is shown in Fig. 19. The characteristics are listed in Table 3. The SRAM unit is 380 µm x 215 µm and the Add/Flag Control unit is 380 µm x 45 µm. In total, the core size is 380 µm x 260 µm. The SRAM organization is 256 words x 16 bits. The fabricated core does not include the SB or the MCU Interface. The added

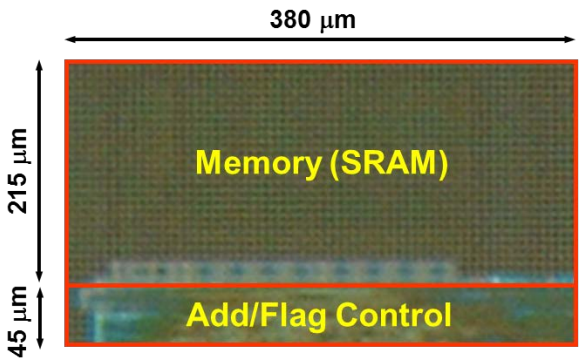


Fig. 19 Photomicrograph of the fabricated FPSM core.

Fig. 20 Count operation waveforms of the 16-ary counter on the fabricated FPSM core.

Table 3 Characteristics of the FPSM core.

Technology	0.18 µm 5 metal CMOS
Core size	380 µm × 260 µm
Organization	256 words × 16 bits
Supply voltage	1.8 V
Frequency	50 MHz
Power dissipation	1.0 mW (@ 50 MHz)

Add/Flag Control unit for the PMU is 1.4 kgates in two input NAND gate, occupying 17.3 % of the core area.

In our design, the SB is 0.98 kgates, which is estimated to be 0.012 mm^2 assuming that the area is proportional to the gate count. Roughly speaking, one SB is provided for one PMU; thus, the area overhead of the added circuits to the memory in the FPSM is 26.3%. This value is slightly high, and overhead reduction is a future issue. The gate counts and area of the MCU Interface are estimated to 4.8 kgates and 0.061 mm^2 respectively. Since the MCU Interface is provided for the FPSM, its overhead is negligible compared to that of an MCU chip.

Fig. 20 shows the measurement waveforms of the 16-ary counter configured on the PMU core for the FPSM. The

Table 4 Comparison of the FPSM and FPGA.

	FPSM			FPGA		
Function	No. of PMUs	Area (mm^2)	Power [1] (mW@20 MHz)	No. of [2] ALUTs	Area [3] (mm^2)	Power [3] (mW@20 MHz)
Free Run Timer (16 b)	2	0.222	0.4	9	0.23	1.12
FIFO [4] (256 w x 8 b)	2	0.222	0.4	19	0.50	2.36

1) Power is independent of the number of PMUs, because only one PMU operates over most cycles.
2) Experimental results on the ALTERA Stratix II.
3) Converted data from the 0.22 μm data on Ref. [11].
4) Data memory is not included.

core operates at a maximum frequency of 62 MHz with a 1.8 V power supply voltage. The power consumption is 1.0 mW at our target frequency of 50 MHz.

The area and power of the FPSM are compared with those of the FPGA using typical peripheral functions, a 16 bit counter (free run timer) and a 256 word x 8 bit FIFO. Table 4 shows a comparison of area and power consumption between FPSM and FPGA. In Table 4, two circuits are selected as the peripheral functions. All functions configured on the FPSM listed up in Table 1, with the exception of calculation units, can be configured using serially connected FFs. Upon increasing the bit width of each function, the number of FFs increases proportionally. This means that fundamentally, the number of PMUs increases in proportion with the bit width. Therefore, if a basic function with a certain bit width can be mapped to the PMUs, the other functions can be deduced. Although the benchmark circuits are not adequately thorough, their results include essential points.

The hard wired logics of the 16 bit counter and 256 words by 16 bits FIFO memory are experimentally mapped on the FPGA, the ALTERA-Stratix II. In Table 4, the number of the ALM, Adaptive Logic module, is our experimental data and is converted to the number of Xilinx CLBs assuming CLB = 2 slices = 2 ALMs, because a Vertex slice is approximately equal to a Stratix II ALM [16], [17]. Under this assumption, the area and the power of ALM are converted from the Xilinx's Vertex data [11] (see the right part of Table II in Ref. [11]).

The number of the used ALMs are 9 and 19, respectively, for the 16 bit counter and the 256 word x 8 bit FIFO. Since the area of the ALM are not known explicitly, we convert the 0. 22 μm data on Ref. [11] to 0.18 μm ones based on the relation mentioned above and the scaling law. According to Ref. [11], the average area and average power consumption per CLB (Xilinx's Configurable Logic Block) are 0.0780 mm^2 and 0.587 mW (Standard deviation is 0.022 mW) at 20 MHz with a power supply voltage of 2.5 V, respectively, in 0.22 μm process technology. Here, we assumed that the density of the ALM is half to the one of the CLB mentioned above. The area of the ALM in 0.18 μm process technology is estimated by multiplying (0.18 μm/0.22 μm)^2 to the average area of the CLB on Ref. [11]. Thus, the hard wired 16 bit counter with 9 ALMs has an area of 0.31 mm^2.

Considering fCV^2 of the power consumption, the 0.22 μm data can be converted to the 0.18 μm data by multiplying (0.18 μm/0.22 μm), and (1.8 V/2.5 V)^2 to 0.22 μm data. The converted data are summarized in Table 4 together with the fabricated FPSM data. In our design, the SB is 0.98 kgates, which is estimated to 0.012 mm^2, assuming that the area is proportional to the gate counts. The fabricated core does not include the SB, while the area of the FPSM in Table 4 includes the area of the one SB per PMU. The FPSM has half the area of the FPGA, and it consumes one-fifth to one digit order less power than the FPGA.

The power consumption does not depend on the number of PMUs, because only one SRAM operates during most of the cycles in the 16 bit counter and FIFO memory, resulting in less power consumption. In the case of the 16 bit counter, when the lower 8 bit counter counts 256 cycles, the upper counter counts one. This means that one PMU operates 256 times, and the other one PMU operates one time during 256 cycles. Thus, power consumption of the upper counter is 1/256 times that of the lower counter, and it is negligible. In the FIFO on the FPSM, one PMU each is assigned to the write address pointer, read address pointer, and data memory. When the FIFO receives the write (read) control signal, the PMU for the write (read) address pointer operates and the write (read) address is outputted in the first cycle. In the next cycle, the PMU for the data memory stores (read out) the data; after which the write (read) operation is completed. This FIFO does not execute write and read operations simultaneously, so one PMU operates during one cycle.

8. Conclusion

We proposed the Field Programmable Sequencer and Memory (FPSM), which is a programmable unit optimized exclusively for MCU peripherals. The FPSM consists of

the Programmable Memory Unit (PMU) array of memory units (Memory) with a small logic unit (Add/Flag Control) controlling the memory address transition cycle by cycle. Four kinds of address transitions are allowed in the Add/Flag Control unit. These address transitions cover almost every peripheral normally required in the field. This limitation simplifies the Address/Flag Control unit and suppresses its area increase. The PMUs not configured for peripherals are used as the standard built-in memory, that is, the FPSM functions both as the memory and as the peripherals. The FPSM is implemented on the FPGA and the programmability is emulated on the FPGA. Furthermore, the FPSM core, one PMU with a 4K bit SRAM, is fabricated in 0.18 μm CMOS process technology with 5 metal layers. The added Add/Flag Control unit is 17.3% of the total area. The FPSM has half the area of the FPGA and it consumes one-fifth to one digit order less power than FPGA. The proposed FPSM could be very useful for future MCU platforms.

References

[1] E. D. Kyriakis-Bitzaros, N. A. Stathopoulos, S. Pavlos, D. Goustouridis, and S. Chatzandroulis, “A reconfigurable multichannel capacitive sensor array interface,” IEEE Trans. Instrumentation and Measurement, vol. 60, no. 9, pp. 3214-3221, Sept. 2011.

[2] I. Adly, H. F. Ragai, A. El-Hennawy, and K. A. Shehata, “Over-the-air programming of PSoC sensor interface in wireless sensor networks,” in Proc. IEEE MELECON, 2010, pp. 997-1002.

[3] F. Hu, S. Lakdawala, Q. Hao, and M. Qiu, “Low-power, intelligent sensor hardware interface for medical data preprocessing,” IEEE Trans. Information Technology in Biomedicine, vol. 13, no. 4, pp. 656-663, July 2009.

[4] J. Xi, C. Yang, A. Mason, and P. Zhong, “Adaptive multi-sensor interface system-on-chip,” in Proc. IEEE Conf. Sensors, 2006, pp.50-53.

[5] N. Aibe and M. Yasunaga, “Reconfigurable I/O interface for mobile equipments,” in Proc. Int. Conf. on Field-Programmable Technology, 2004, pp.359-362.

[6] S. Tanaka, N. Fujita, Y. Yanagisawa, T. Terada, and M. Tsukamoto, “Reconfigurable hardware architecture for saving power consumption on a sensor node,” in Proc. Int. Conf. Intelligent Sensors, Sensor Networks and Information Processing, 2008, pp. 405-410.

[7] A. Doboli, P. Kane, and D. Van Ess, “Dynamic reconfiguration in a PSoC device,” in Proc. Int. Conf. Field-Programmable Technology, 2009, pp. 361-363.

[8] V. Baumgarte, G. Ehlers, F. May, A. Nückel, M. Vorbach, and M. Weinhardt, “PACT XPP - A self-reconfigurable data processing architecture,” The Journal of Supercomputing, vol. 26, issue 2, Sept., pp. 167-184, 2003.

[9] T. Sato, H. Watanabe, and K. Shiba, “Implementation of dynamically reconfigurable processor DAPDNA-2,” in Proc. IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test, 2005, pp. 323-324.

[10] H. Amano, S. Abe, Y. Hasegawa, K. Deguchi, and M. Suzuki, “Performance and cost analysis of time-multiplexed execution on the dynamically reconfigurable processor,” in Proc. IEEE Symposium on Field-Programmable Custom Computing Machines, 2005, pp. 315-316.

[11] K. Nakamura, T. Sasao, M. Matsuura, K. Tanaka, K. Yoshizumi, H. Nakahara, and Y. Iguchi, “A memory-based programmable logic device using look-up table cascade with synchronous static random access memories,” Japanese Journal of Applied Physics, vol.45, no.4B, pp.3295-3300, Apr. 2006.

[12] Y. Kawamura, “A reconfigurable microcomputer system with PA3 (Programmable autonomous address-control-memory architecture),” in Proc. IEEE Asian Solid-State Circuits Conference, 2007, pp. 388-391.

[13] S. V. Kartalopoulos, “Linear dynamic feedback sequential machines with matrix memory implementation,” in Proc. Decision and Control, 1982, pp.1233-1241.

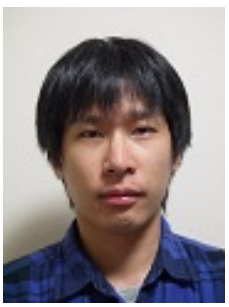
[14] L. D. Coraor, P. T. Hulina, and O. A. Morean, “A general model for memory-based finite-state machines,” IEEE Trans. Computers, vol.C-36, no.2, pp.175-184, Feb. 1987.

[15] L. Gerbaux, and G. Saucier, “Automatic synthesis of large Moore sequencers,” in Proc. European Conf. Design Automation, 1992, pp. 237-244.

[16] https://www.altera.com/content/dam/alterawww/global/en_US/pdfs/literature/hb/stx2/stratix2_handbook.pdf, “Stratix II Device Handbook, Volume 1,” p. 2-7.

[17] http://www.xilinx.com/support/documentation/data_sheets/ds003.pdf, “Virtex 2.5V Field Programmable Gate Arrays, Product Specification,” p. 5.

Yoshifumi Kawamura was born in 1960 in Miyagi, Japan. He graduated from the Electrical Engineering of Miyagi Technical College. In 1981, He joined the Semiconductor & Integrated Circuit Division on Hitachi Ltd. and was engaged in the development of telecommunication devices. In 1993, he was transferred Graphics Communication Laboratories and was engaged in the research and development of MPEG2 multimedia communication systems for four years.In 1997, he was engaged in development of LSI’s for GSM Cell phone. In 2003, He was transferred Renesas Technology Corporation, next Renesas Electronics Corporation, and was engaged in the research and development of programmable and reconfigurable IPs. He is now with SKY technology Co., Ltd.



Naoya Okada was born in Fukui, Japan. He received the B.S. degree in electrical and electronic engineering and the M.S. degree in electronic information engineering from Kanazaw University. He is now with the Information Technology R & D Center, Mitsubishi Electric Corporation.

Yoshio Matsuda was born in Ehime, Japan, on October 26, 1954. He received the B.S. degree in physics and the M.S. and Ph.D. degree in applied physics from Osaka University, Osaka, Japan, in 1977, 1979, and 1983, respectively.
He joined the LSI Laboratory, Mitsubishi Electric Corporation, Itami, Japan, in 1985. He was engaged in development of DRAM, advance CMOS logic, and high frequency devices and circuits of

compound semiconductors. Since 2005, he has been a professor at the College of Science and Engineering, Kanazawa University, Japan. His research is in the fields of integrated circuits design where his interests include multimedia systems, low power SoCs, and image processing LSIs.

Tetsuya Matsumura received the B.E. and Ph.D. degrees from Kyushu Institute of Technology, Fukuoka, Japan in 1984 and 2001, respectively. He joined Mitsubishi Electric Corporation in 1984. He was transferred to Renesas Technology Corporation, Renesas Electronics Corporation, in 2003 and 2010, respectively. Since 2013, he has been a professor of department of computer science, college of engineering at Nihon University, Fukushima, Japan. His research is in the fields of next generation multimedia system.

Hiroshi Makino was born in Osaka, Japan, in 1959. He received the B.S. degree in physics from Kyoto University, Kyoto, Japan, in 1983 and the Ph.D. degree in electrical engineering from the University of Tokyo, Tokyo, Japan, in 1997. In 1983 he joined the LSI R&D Laboratory, Mitsubishi Electric Corporation, Itami, Japan, where he worked on the research and development of GaAs digital LSIs until 1990. From 1991 to 2002, he was engaged in the research and development of Si CMOS high-speed and low-power digital circuits in System LSI Laboratory and System LSI Development Center. From 2003 to 2007, he continued the same study at Advanced Design Framework Development Department of Renesas Technology Corporation, Itami, Japan. In 2008, he has become a professor at Osaka Institute of Technology, Osaka, Japan, and is working on the research and education of system LSI design.

Kazutami Arimoto received the B.S., M.S., and Ph.D. degrees in electric engineering from Osaka university, Osaka, Japan, in 1979, 1981, and 1993, respectively. He joined the LSI Laboratory, Mitsubishi Electric Corporation, Itami, Hyogo, Japan, in 1981.Since then, he has been engaged in the design and development of DRAMs and IPs for System LSI. He transferred to Renesas Technology Corporation, Renesas Electronics, and Okayama Prefectural University, in 2003 in 2010 and 2012, respectively. Currently, he focused on image processing IPs, communication IPs, sensor interface, memory based IPs and re-configurable IPs for multimedia, security, auto mobile, network, sensor network, energy management, and future intelligent embedded systems.

Prof. Arimoto has 192 U.S. patents and 69 Japanese patents issued. He is a senior member of the Institute of Electronics, Information and Communication Engineering (IEICE) of Japan. He is also a TPC member of ISSCC and A-SSCC.