I J C T A, 9(2) 2016, pp. 1155-1168
© International Science Press

Time and Power Reduction of Embedded DSP Systems using Register-program Execution

D. Haripriya[*], C. Govindaraju[**] and M. Sumathi[***]

ABSTRACT

In the embedded systems large power savings can be achieved through reduction of transition in the busses. Embedded application's program memory cost significant portion in the power dissipation profile. In this paper a method to reduce the address and data bus transition is proposed that will significantly reduce the power spent on program memory. We propose a register-program execution, which controls the switching activity, and reduce the power and energy spent on program fetching. The proposed scheme mechanism is applied on the DSP core TMS320c54x and evaluated for the enhanced instruction fetch energy using standard DSP benchmarks.

Keywords: Digital Signal Processing, Bus operation, Embedded Systems, Fetching, Pipeline, Energy.

1. INTRODUCTION

The battery powered, handheld devices serve as access point for hardware infrastructure for today's digital world. Yet the cost and short battery life trim down their enormous potential. The cost can be reduced with economic factors, while the battery life management needs special mechanisms to provide better performance.

Embedded devices are developed for application specific operation. When considering the embedded devices the general design is aimed at to have light weight and to produce less heat along with application specific features. The ergonomics and ease of use increase with reduced weight and less heat production. When the design is optimised for power, as well as energy the above two goals can be realised. Any general purpose or embedded computing device can be improved to be small in size by reducing the size of the battery and arangments for heat sink.

Equipment designed especially for embedded applications can be optimized for low power consumption mainly by two appraoches. The first is changing the processor's circuitery to consume low power or energy. Tuning the processor archetecture towards low power operation give good results in terms of power. The second method being optimization of the program for any given processor.

From the standpoint of processor design, a number of techniques have been used to reduce power usage. These techniques can be of two types. Either the processor's circuitry can be designed to use less power or, the processor can be designed in a manner that permits power usage to be managed on the fly.

On the other hand, given a particular processor design, its programming can be optimized for reduced power dissipation. Thus, from a programmer's standpoint, there is often more than one way to program a processor to perform the same function. For example, algorithms written in high level programming languages can be optimized for efficiency in terms of time and power. Until recently, at the assembly language level,

[*] Department of Electronics and Communication Engineering Jawaharlal Nehru Technological University, Kakinada.
[**] Government College of Engineering, Salem.
[***] Sathyabama University, Chennai
E-mail: dharipriya38@gmail.com

most optimization techniques have been primarily focused on speed of execution without particular regard to power use.

On the other hand, given a particular processor design, its programming can be optimized for reduced power dissipation. Thus, from a programmer's standpoint, there is often more than one way to program a processor to perform the same function. For example, algorithms written in high level programming languages can be optimized for efficiency in terms of time and power. Until recently, at the assembly language level, most optimization techniques have been primarily focused on speed of execution without particular regard to power use.

In embedded applications, program execution in the chip consumes a significant part of the entire energy from the battery. Hence, reducing the execution time of the program will in turn increase the battery life. Generally to reduce the software development time and also due to the increased complexity of the embedded software, embedded system programmers use high level languages. This, in turn, results in the use of compiler-generated code in embedded applications that is much larger than hand-optimized code, putting additional pressure on the instruction fetch stage. Several techniques to reduce flips in the address and data bus gained considerable attention in embedded applications. A few examples of such techniques are gray coding in address bus [13] and instruction pairing [1].

This paper focuses on reducing the pipeline stages, and hence the execution time, as well as the power consumed in an embedded system. We concentrate on embedded systems, which use processor cores, typically Digital Signal Processors (DSPs). DSP architecture is optimized for the digital signal processing applications. For DSP circuits the biggest and most active switching capacitances usually consist of the global address and data buses. There are several approaches in minimizing the dissipated power on these lines by reducing the voltage swing [5] or recovering the injected energy with adiabatic circuit techniques [6]. Another idea is to reduce the switching activity on these buses by using alternative number representations [3].

In this paper, we propose a common hardware mechanism, referred to as the Register program scheme, which can be used for both pipeline stage reduction and power reduction. The scheme is based on a register-program table using which instruction are stored and executed with in CPU to reduce fetch stage and power. The proposed register-program scheme is simple and can easily be fitted in the pre-decode stage without incurring any additional runtime overhead. Also, the proposed scheme can be used for either normal execution or register-program execution. Further, our scheme achieves power reduction by change the state of both address and data bus to high impedance state and also avoid the first two stages in the pipeline. Reducing the instruction fetch. Reducing the power consumption of instruction fetch operation stage is important as the instruction fetch stage is known to contribute more than 15% of the total power consumed [11]. We evaluate the proposed Register-program scheme using standard benchmark programs. The instruction Register-program scheme achieves reduces the execution time over 30%. The scheme also results in an energy reduction of more than 82% of the instruction fetch power.

The rest of this paper is organized as follows. In Section 2, we present a discussion on related work. We describe the proposed register-program scheme in the subsequent section. In Section 4, we present the details of a register program table organization and explain how runtime execution can be achieved using the register-program table. The elimination of pipeline stages due to register-program execution are presented in Section 5. In Section 6, we discuss how the Register-program scheme can achieve reduction in instruction fetch power. This section also reports the energy reduction in three standard benchmark program embedded control applications. Finally we provide concluding remarks in Section 7.

2. RELATED WORK

In this section we discuss related work in execution time reduction and power reduction in the real time application areas of the Embedded DSP processor.

A 460 MHZ at 397mv,2.6GHZ at 1.3v,32bits VLIW DSP Embedding F_{max} Tracking[20] experiments about bringing Energy efficiency in the mobile Applications.In order to achieve lowest peak maximum frequency tracking design technique is used. The proposed DSP is used for the telecom applications. UTB-FDSOI technique is used for the high performance of the processor at ultra low voltage real time applications. The processor architecture gives a very good performance when forward biasing up to 0 to 2v. However, applying a forward negative voltage results in the leakage current. But upon lowering the input voltage to 1volt results in dynamic power is considerably reduced.

Energy Efficient Digital signal processing [21] analyses about the DSP tms320c55x being used in the battery operated devices for area and energy efficient digital signal processing. As a whole, the the system level tasks are changed in to silicon level. The idea is analysed with the hearing devices and in the medical imaging .

Design of low voltage block and ADC for energy –efficient systems [22], the paper mainly concentrates on designing different analog and digital blocks, memories to operate at very low voltages itself. The paper concludes with an idea that, though the circuits and devices can be operated at low voltages for energy consumption. In circuit and architectural level different technologies can be incorporated for highly energy efficient system design.

A Low power processor with configurable Embedded machine-Learning accelerators for hign-order and adaptive analysis of medical sensor networks [23], the papers says about the low power physiological signal measurement for the biomedical applications. The system is designed with vector machine accelerators to under stand the algorithms.The embedded vector algorithm for active learning and to adapt for the patients specifc signals.The on chip processor helps to minimize the human efforts.

Real time Instruction-cycle –Based dynamic voltage scaling power management for low power digital signal processor with 53% energy saving [24], Tun-Hao Yu et.al discusses about the instruction cycle based voltage scaling making use of DVS algorithm.The chip is implemented on the HH-NEC 0.18 μm CMOS process.

Improving energy efficiency in FPGA through judicious mapping of computation to embedded memory blocks [25], this idea is to implement an FPGA based system for DSP memory blocks in order to reduce the power and the area in real time DSP applications.

A Hardware –efficient variable Length FFT processor for Low power Applications[26], This paper discusses about the energy and area reduction of the embedded DSP appllcations.Also, it makes use of the trounding and scaling methods are used to improve the signal to quantization noise ratio. The word length is reduced in such a way to have a minimum power is spent and the life time and the system is increased. The FFT processor was designed with verilog HDL and the synthesis is done using synopsys's design compiler.

Application of MSP430 and C5000 in mobile patient and environmental monitoring[27], in this work two sensors are used. One is to measure the respoiratory sounds and the other is to find the environmental pollution, where the patient is present.Also, the system is designed only at the prototype level.

Real-time bilateral filtering of ultrasound image through highly optimised DSP implementation[28], the paper experiments on the real time implementation of bilateral filter in DSP using tms320c64x+.

2.1. Execution Time And Power Reduction

A register based low power processor architecture using a decoded instruction buffer is presented in Hiraki *et al.* [7]. In this approach, a small RAM is designed for storing the decoded instructions temporarily inside the processor during program execution and is named as decoded instruction buffer (DIB). From the second iteration of a loop, the first two stages of the pipeline namely the instruction fetch and decode are stopped and the decoded instructions are fed directly from the DIB. A considerable power saving is possible by this approach because, execution of tight loops take substantial amount of the execution time spent on them in many signal processing applications. Around 40% reduction in power consumption was obtained using this method. But this method is useful only for a ‘for loop’ like structures since it cannot handle deviations from the loop during execution. This method cannot be applied when the program sequence gets beyond the loop address limits based on some decision making steps inside the loop which usually happens in real time signal processing systems.

It also cannot work for jumps within the loop structure based on conditions at run time. Hence this technique may not be useful for all real time embedded applications. Where separate counter is used for maintaining a loop count, it will also consume some power. Our approach aims at optimizing the power spent on instruction fetching with the natural flow of the program taken into account. It may be noted that the decoded values of instruction takes more space for storing. For an embedded application, this causes an unwanted cost which is avoided in our method.

Another instruction fetch energy reduction method by loop caches is presented by Lee et al., in [9]. This approach is similar to the approach discussed in [7]. Both the approaches attempt on reducing the execution energy of the tight loops. In this approch, a small cache called loop cache is implemented which is used for storing the recently fetched instructions. A new instruction called “sbb” identifies the small loops and cuts off the main cache while the instructions are executed from the loop cache. Because of this the fetch energy is reduced due to minimized main cache access. The results are reported to have a 40% reduction in the main cache access. The actual energy reduction achieved is not presented in this paper.

Subash *et al.,* discuss an instruction re-map table using SRAM structure [8]. This approach uses an Instruction Re-Map Table (IRT) which acts as a decompression unit for compressed instructions. The repeated segments of the program are encoded into the IRT and are referred as the address if their location in the IRT. But the encoded compressed instructions are represented by 17 bits wide in the program memory instead of storing them as 16 bits. Since the seventeenth bit only can differentiate the compressed and uncompressed instructions, it is needed to have an extra in their program memory for the instructions. This approach reduces the flips in the program memory data bus to an extent but not completely. The results indicate up to 40% reduction in power consumption. This approach changes the entire program memory as 17 bits wide and increases the size of the memory required which is not desirable for embedded applications.

3. REGISTER-PROGRAM SCHEME

This scheme we propose uses a register structure which is inside the CPU. The segments of frequent occurrence will be loaded into the Register Program Table (RPT) and will be executed from there instead of from the program memory. This RPT can be programmed with the extra instructions which are explained in the section 4.3. This configuration can be done either once prior to running a given application (Pre-configured execution), or possibly many times while running a given application program (Run-time configured execution).

3.1. Pre-Configured execution

In the pre configured mode the entire program is configured once inside the RPT. This is possible for only the small embedded applications which will be less than the number of locations in the RPT. Since many of the embedded applications are of infinite loop based program, this configuration may be used to store the single largest loop inside the IRT.

3.2. Runtime Configured Execution

When the application program has many loop structures, this mode of operation is adopted while writing the application. The segment is configured in the RPT dynamically with the help of the special instructions. While running the program, these instructions are executed from the RPT.

It may be noted here that in both the Pre-configured and Runtime configured modes the reconfiguration segments, the number of instructions involved, their starting points and ending points are decided during the program design phase. The actual configuration happens dynamically at run-time. This happens once for the Pre-configured mode and multiple times for the Runtime configured mode. In the following section we discuss how the rewriting of instructions to registers is done and reduction of pipeline stages is achieved.

4. REGISTER-PROGRAM TABLE

To rewrite the into the CPU the RPT is used as an internal register (a RAM like structure). The RPT can be viewed as a general-purpose register (normal CPU register). The register of 16 bits wide and stores its contents from the program memory. The size of the RPT decides the maximum possible length of loop operation inside an application. For any size of the RPT the instructions are referred using normal Program Controller for execution. Four special instructions are added to the existing instruction set for the reconfiguration of the RPT and to control the bus activity.

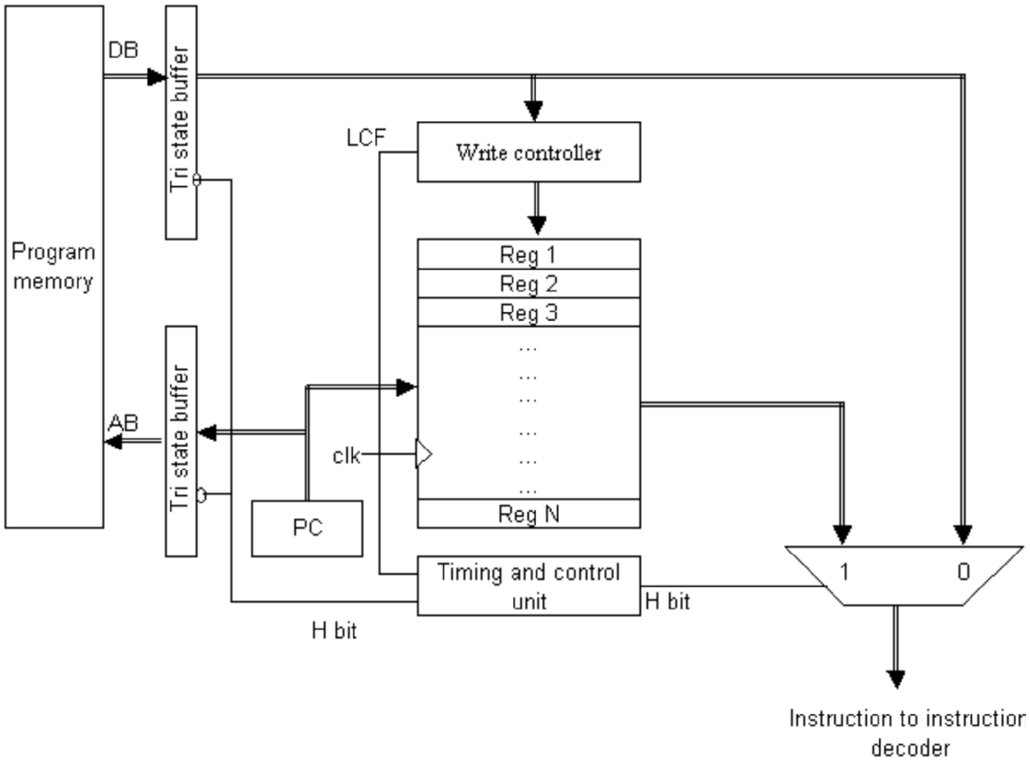


Figure 1: Register-program table

4.1. Design of Register-Program Table

The instructions that are written into the Register Program Table are same as the content of the program memory. We create a small block of image in the CPU by the help of the RPT. The size of the reconfigured block inside the RPT is same as that of the original block size in the program memory. The loop based structures are identified in the program and selected to be configured inside the RPT. The method of configuring the RPT is explained in the following sections. After rewriting the instructions in the RPT, each of the register holds a normal 16-it instruction.

If the RPT is designed to have a total of 256 locations, the location address of the instruction stored inside the RPT is same as the lower eight bits of the instruction's original program memory address. Since the address of any instruction is same as the Program Counter's normal address lsb[0:7], it can be accessed by the Program Counter's address itself. Once an instruction is written into the RPT, it can be referenced by the address of its location. Since the width of the address, is the program counter lsb[0:7] value.

If N is the number of locations inside the RPT then, the number of address bits necessary to locate the instructions is log2N. The figure 1 shows the structure of the RPT. The (Reg1), (Reg2) etc., are the locations present inside the RPT with the addresses addr1, addr2 respectively, where these addr1, addr2 etc., are only last eight bits of the Program counter. The higher order bits of the address bus are not used. The same Register Program Table can be used for accessing any address location of the program memory address bus. (Program memory (A0:A7) = Register-program table (a0:a7)).

4.2. Implement of Register Program Table

To explain the integration of the Register Program Table to the processor architecture, the Texas Instruments DSP core TMS320c54x is used as a representative architecture. The TMS320c54x DSP core has six pipeline stages, viz., Initiate-Fetch, Complete-Fetch, Predecode, Decode, Initiate-Read, Complete-Read, Execute and Write stages.

The first stage of the pipeline is the istruction fetch stage which takes care of placing the address of that particular instraction in the address bus. During Complete-Fetch stage, the instruction is available in the program data bus from the memory. The two stages called predecode and decode take care of decoding the complete instruction.

The execute stage does the instruction and the results are stored the write back stage of the pipeline. This core can be enhanced by inclusion of the Register Program Table. The Register Program Table fits in the pre-decode stage of the pipeline as indicated in Figure 2.

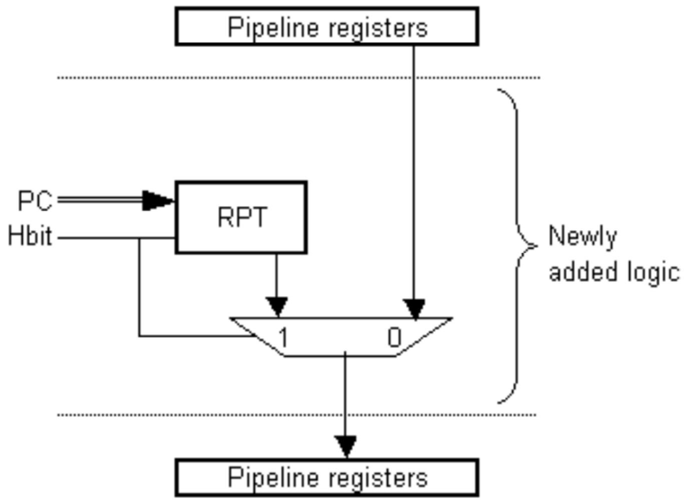


Figure 2: Modified decoded stage

The access to the RPT is same as the access to the general purpose register inside the CPU. The time of access is very small and is equivalent to the SRAM module time which can fit in the pre-decode stage.

4.3. Configuring the RPT

In this section we discuss how to configure the RPT. We introduce two pairs of instructions SEGST, SEGED, and HBUS, RBUS. The first pair of instruction is used to copy most repeated segment program that is, the loop structure into the RPT register. The second pair of instruction is used to choose the execution of the instructions from program memory or RPT by way of controlling the address bus state of the program memory.

The instruction SEGED stands for "segment start" and SEGED stands for "segment end". In the second pair HBUS stands for "High impedance the bus" and RBUS stands for "release the bus." The semantics of these instructions are described below. The registers SEREG is a register used for storing the Program Counter's content during the operation.

SEGST: This instruction is decoded by the instruction decoder in the CPU control unit. On execution this will set loop control function bit (LCF) and copy the next program counter value to the SEREG. The loop control function bit operates the CPU just like MOV operation for memory to register. The next instructions are fetched from the consecutive locations in the program memory. The control unit controls the write controller of the RPT to write the instructions inside. The address bus of the RPT is connected to the program address bus which is handled by the Program Counter(PC) . The location selection uses only the lower order PC value from lsb0 to lsb7. So sets of consecutive locations in the RPT are written with the instructions that are fetched from program memory. This rewriting happens till an SEGED instruction is fetched.

SEGED: This instruction is decoded by the instruction decoder and makes the CPU control unit copy the SEREG value to the Program Counter. This instruction also resets the loop control function bit. Since SEREG content is sent to the PC, now the execution of program shifts to the next instruction of SEGST in the program memory. If the loop control bit is set the SEGED instruction is consider as NOP operation.

HBUS: This instruction is decoded by the instruction decoder and the CPU control unit changes the program memory address bus and data bus to high impedance state and sets the bit H equal to 1.

RBUS: This instruction is decoded by the instruction decoder and the CPU control unit releases the bus and sets the bit H equal to 0.

The sample program memory and register-program table content shown in the fig. The SEGST instruction copies the content of program in program memory to Register Program Table. This instruction enables the repeated sequence instruction are executed with out program memory and buses. The segmented instructions are copied in registers the sequence of program execution again transferred to the next instruction of SEGST that is HBUS instruction. The instruction HBUS is now executed the bus are now changed to high impedance state so that after this instruction the program memory is not accessible, the next instruction are executed in the register-program table only. In this case both the fetch pipeline stages are skipped the CPU now running in high performance and economic power mode.

4.3.1. Bus Control in Configured Execution

The control and operation of the address and data buses of program memory during register Configured operation is discussed here. When a branching like instruction is used in the program execution, the CPU signals coordination should be designed carefully. This is illustrated with the help of an example. Figure 4 shows the segmented code with more than one repeated sequence instruction with different segments. The Target1 is reachable by a branch instruction from code segment 2 and code segment 3. Here the section starting from the location Target1 up to the SEGED form a loop which we call as Loop1 in the original program and is configured inside the RPT. Since the bus is in the high impedance mode the access will be from the RPT from the second iteration onwards.

Inside the Loop1 if there are branching conditions present whose target addresses are inside the Loop1 itself, then the configuration of RPT for loop1 can continue without any need to come out of the Loop1. Any segment like segment 1, segment 2 and segment 3 can call and use the Loop1 and release the HBUS

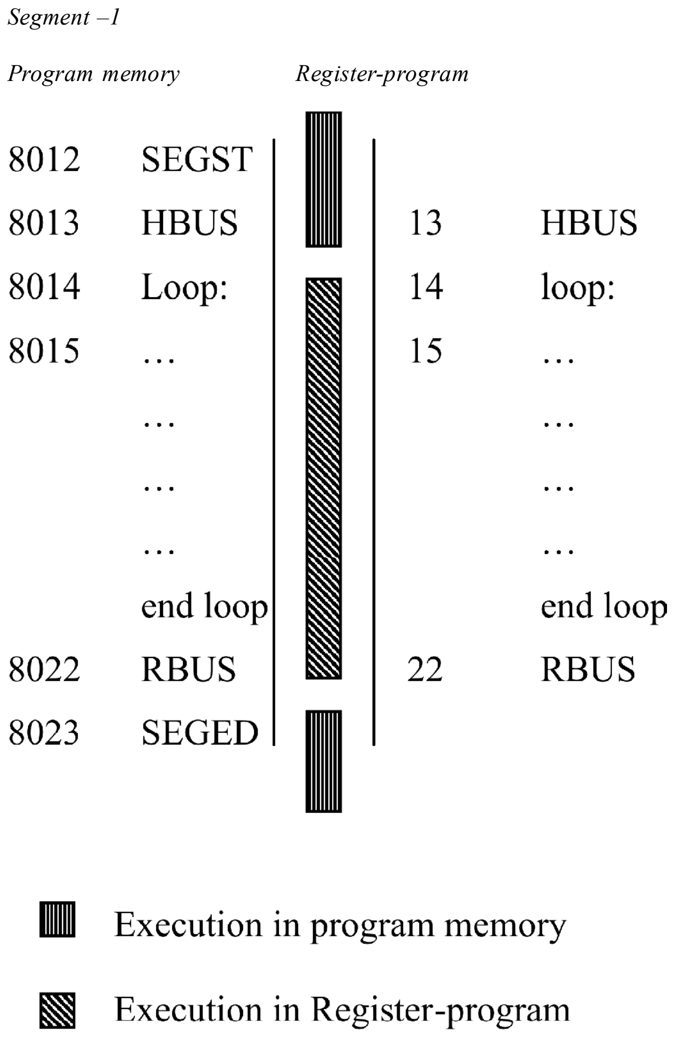
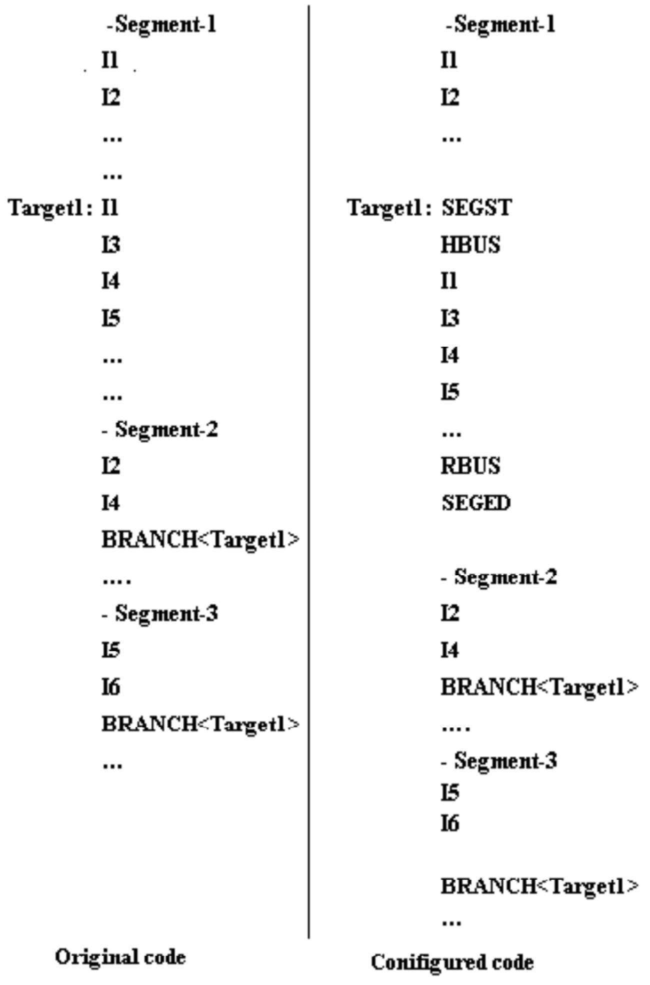


Figure 3: Register-Program execution



```
          -Segment-1                      -Segment-1
          I1                              I1
          I2                              I2
          ...                             ...
          ...
Target1:  I1                    Target1:  SEGST
          I3                              HBUS
          I4                              I1
          I5                              I3
          ...                             I4
          ...                             I5
          - Segment-2                     ...
          I2                              RBUS
          I4                              SEGED
          BRANCH<Target1>
          ....                            - Segment-2
          - Segment-3                     I2
          I5                              I4
          I6                              BRANCH<Target1>
          BRANCH<Target1>                 ....
          ...                             - Segment-3
                                          I5
                                          I6

                                          BRANCH<Target1>
                                          ...
      Original code               Conifigured code
```

Figure 4: Branching from different segments

after the loop execution. Likewise any number loops can be dynamically configured and taken back to and from the RPT.

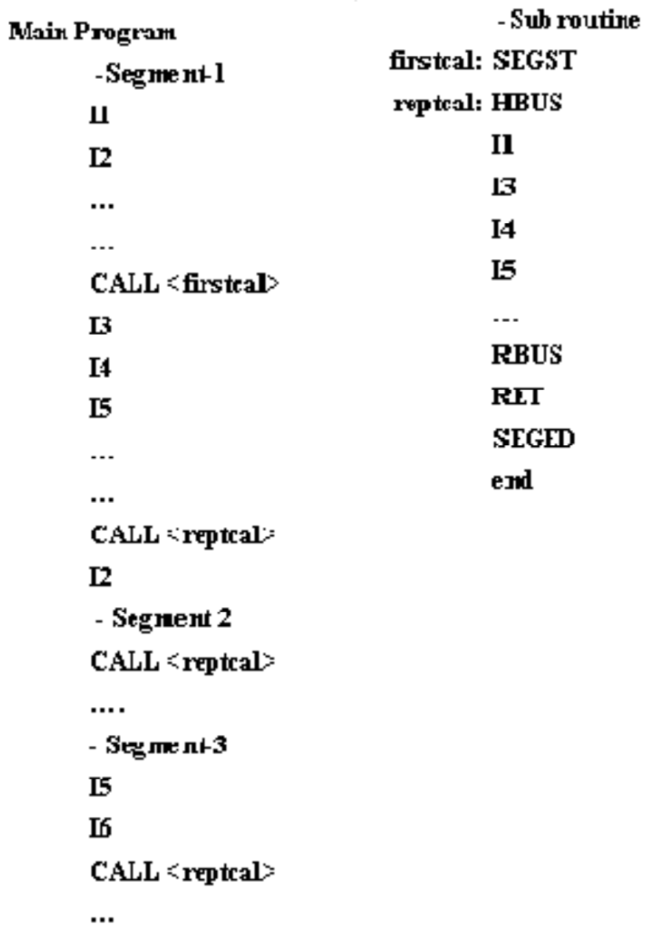
One more advantage of this SREG and RBUS pair is the when the same Loop1 is called by many calling instructions inside the whole program, we can give the address of the HBUS as the target called address instead of the original SEGST address. If the program has only one loop or if the program flow is predicted and made sure about which loop is presently inside the RPT by common flags, this method can be used. Since the address of HBUS is given for the repeated calls, the repeated configuration time and energy is saved for the whole program execution. The following Figure 5 shows the example of repeated calling without configuring the RPT.

5. EXECUTION TIME REDUCTION

In this section, we discuss the execution time reduction achieved by pre-configured execution and runtime-configured execution methods.

5.1. Pre-Configured Execution

In this scheme, we have written the instructions once, prior to running a given application with the objective of reducing the pipeline stages so that reduces the overall execution time for that particular application. The



```
Main Program
        -Segment-1
        I1
        I2
        ...
        ...
        CALL <firstcal>
        I3
        I4
        I5
        ...
        ...
        CALL <reptcal>
        I2
        - Segment 2
        CALL <reptcal>
        ....
        - Segment-3
        I5
        I6
        CALL <reptcal>
        ...
```

```
           - Sub routine
firstcal:  SEGST
reptcal:   HBUS
           I1
           I3
           I4
           I5
           ...
           RBUS
           RET
           SEGED
           end
```

Figure 5: Branching with repeated calls

frequently used segments are written in to the register-program table at the time of beginning of the execution start.

For smaller application the entire program is completely transferred to the Register Program Table In this state the embedded applications entire lifetime working reduces two pipeline states that is without initial fetch and complete fetch operation. Here the application program size should be lesser then the Register Program Table size. The size of register program table depends on the accessing width line. The width i.e., 2,3,4,8,9…16 coressponding register program table size 4,8,16,256,512…. .for practical reason we use register program table size is 1024 so that for accessing the register in the register-program table we use program counter lsb[0:9] only.

We study the effect of reduction of pipeline stages and in turn reduction of execution time of the application program with different size of bench mark programs. Table sizes beyond 2,048 are analyzed, as the improvements achieved are very high for the frequent switching of between the loops waiting for real-time environment signal. The reduction of pipeline stages is expressed in reduction of execution time

% Time Reduction

$$= \frac{Original\,time - Reduced\;pipeline\,execution\,time}{Original\,time} * 100$$

In all our experiments, the original code considered is a highly optimized code.

Table 1
Execution time reduction in pre-Configured Execution

Number of Register	*Percentage of execution time reduction*		
	RTA-1	*RTA-2*	*RTA-3*
128	31	22	15
256	31	28	22
512	31	31	28
1024	31	31	31
2048	31	31	31

5.2. Runtime-Configured Execution

In this scheme, the frequently used segments are written on register-program table while running a given application program dynamically.

The application program is divided into several segments based upon the frequency of the segment usage. Each repeated segment size is clearly analyzed which is suitable to the register-program table. The frequently used repeated segments are started with SEGST and ended with SGEND, between the two instruction segment code are written in to the RPT from the program memory.

Table 2
Execution time reduction in runtime-written execution

Number of Register	*Percentage of execution time reduction*		
	RTA-1	*RTA-2*	*RTA-3*
128	18	22	14
256	21	26	19
512	28	28	29
1024	31	31	30
2048	31	31	31

5.4. Access Time of Table Register-Program Table

We evaluate the access time overheads introduced by the Register-program table using the CACTI simulator [19]. CACTI is a cache access cycle time model which is an analytical model based on transistor and wire characteristics derived from SPICE simulations. We use the CACTI simulator to obtain quickly the access time estimates for the Register-program table. The critical path for accessing the Register Program Table is identical to any SRAM structure and primarily consists of decode, wordline, bitline and the sense amp stages. Hence, we measured the latency of the Register-program table using CACTI 3.2. The access time for various Register-program table sizes and feature sizes are given in Table 3.

Table 3
Access time of register-program table

Number of Register	*Feature size*		
	0.18μ	*0.13μ*	*0.09μ*
		(Access time in nanoseconds)	
64	0.68	0.42	0.28
128	0.69	0.44	0.31
256	0.72	0.49	0.33
512	0.76	0.50	0.36
1024	0.81	0.53	0.38

The above access time estimates assume that each entry is of size 2 bytes, As can be seen from the table, for a 0.13_technology, the access time of a 128-entry Register-program table is less than 0.4 ns.. Hence we expect that the Register-program table access time can easily be included in the pre-decode stage without incurring any additional penalty.

6. REGISTER-PROGRAM FOR POWER REDUCTION

In any DSP processor, the instruction fetching stage takes significant portion in the overall power consumption. The primary contributor to the instruction fetch power is the toggling of high capacitance nets that connect the CPU and the memory. The voltage change on a gate capacitance requires charge transfer, and therefore causes power consumption. The toggle in program memory data bus is too high hence the power consumption on data bus high. Data buses are generally fabricated as a set of connections routed together. Each of these lines has a characteristic capacitance with respect to the silicon substrate on which they are fabricated. Since these bus lines frequently are routed adjacent to each other, they also possess an intersignal capacitance, or a capacitance between the adjacent bus lines. When the voltage on a bus line changes, these capacitances also become charged and discharged as described previously. Since some of these capacitances exist between signal lines, the necessity to charge or discharge depends on the voltage levels on both lines.

The toggles in the program address bus are due to the changes in PC values used in fetching the instructions, while the toggles in the program data bus are due to the instructions fetched. The toggles in the program data bus are likely to be higher and contribute significantly to the instruction fetch power. To establish this, we measure the toggles in program address and data buses in three slandered benchmarks. These three benchmarks, referred to as RTA-1, RTA-2, and RTA-3, are same as used for execution time reduction. The distribution of toggles on program addresses and program data buses for these benchmarks is reported in Table 4. From Table 4, it can be seen that program data bus contributes to over 80% of the total number of toggles. As per the Calculation of TMS320LC54x Power Dissipation Application Report spra164 Texas Instruments [4] The worst case (AAAAh/5555h consumes more current than driving 0000h/FFFFh) the total current consumed on program memory buses are 2.88 mA per MHz of bus switching frequency at 3 V. in a embedded application the program execution is endless.

Table 4
Toggle distribution on program address and program data buses

Benchmark	*Percentage contribution to total number of toggle*	
	Program address bus	*Program data bus*
RTA-1	22.5	77.5
RTA-2	17.5	82.5
RTA-3	22.6	77.4

6.1. Pre-Configured Execution

In this scheme, we have written the program memory instructions to registers in the register program table once prior to running the application. The frequently used segments are written in to the register-program table at the time of beginning of the execution start. If the target program size is below the size of Register Program Table the entire target program is written to the register-program table. In this case the application program runs on entire lifetime with out fetching pipe line stages. The power wasted in the conventional pipeline is saved in our register program execution scheme by changing the status of the program memory bus completely to high impudence status for entire lifetime and avoiding the fetching stages. In pre-written execution the RPT is written only once during the whole operation. Due to this, the number of bits fetched reduced only one time. The number of flips in the buses is determined using a developed C program that computes the sum of toggles between adjacent instructions fetched for the entire sequence. The computed result shows 90% flips reduced in the buses 10% is only initial written to the register table. The flips in the buses are shown in the table 5.

Table 5
Flips reduction in pre-written execution

Number of Register	*Percentage of toggle reduction*		
	RTA-1	*RTA-2*	*RTA-3*
128	62	48	33
256	78	62	56
512	91	72	76
1024	91	84	78

6.2. Runtime-Confugured Execution

In pre-written execution the entire target application program is run in register-program table, but if the application program size is too big compared to the register-program table we use the runtime-written execution. In case of run-time written, partition the application program based upon the frequency of sequence loop into segments. Each repeated segments are enclosed with the special pair of instruction. In this scheme the fetching operation is avoided in the segmented loop execution already explained ref fig.2. in run-time written the flips distribution in the busses are shown in the table 6.

Table 6
Flips reduction in runtime-written execution

Number of Register	*Percentage of toggle reduction*		
	RTA-1	*RTA-2*	*RTA-3*
128	58	32	26
256	65	45	49
512	71	66	65
1024	80	74	72

6.3. Energy Saving

The address and data bus require an amount of current that is proportional to the overall bus-switching rate [4]. So the worst case switching current for the entire data bus is 0.09 mA per MHz per bit of data bus switching frequency at 3 V and the worst-case switching current for the address bus is 0.09 mA per MHz per bit of switching frequency at 3 V. the total current consumption for the entire program memory bus is 4.32mA per MHz.

Energy measurements are made as follows. We obtain the switching activity information on the CPU program memory interface and on the register-program execution using flips finding program. Energy reduction for pre-written and run-time written for different table sizes are summarized in Table 15. We compute the energy reduction for the new scheme that resulted in best toggle rate. The current consumption in the program memory bus are then multiplied with the simulation time to get the exact energy consumption.

We observe that for pre-written execution the register size is 256 locations is good enough to yield energy savings of over 82% for RTA -1. With larger register location used for large size program size, there is a achieved very good energy reduction. With run-time execution, we are able to achieve energy reduction of 80% for RTA-2. Due to this high number of runtime written, the energy overhead due to copying instruction to the register-program table is high. The energy saving is very high due the embedded application entire life time working in register-program execution the overhead energy is negligible.

Table 5
Energy spent in pre-written execution

Register Size	*Energy spent (Normalized)*		
	RTA-1	*RTA-2*	*RTA-3*
0	1	1	1
128	1	1	1
256	0.18	0.8	0.9
512	0.20	0.34	0.64
1024	0.22	0.28	0.31
2048	0.25	0.32	0.26

Table 6
Energy spent in Runtime-written execution:

Register Size	*Number of runtime-written*			*Energy spent (Normalized)*		
	RTA-1	*RTA-2*	*RTA-3*	*RTA-1*	*RTA-2*	*RTA-3*
0	0	0	0	1	1	1
128	22	36	45	0.45	0.56	0.75
256	10	26	31	0.26	0.45	0.68
512	1	8	16	0.20	0.35	0.54
1024	1	1	6	0.22	0.28	0.31
2048	1	1	1	0.26	0.32	0.26

7. CONCLUSION

We proposed a mechanism register-program table which to reduce the power consumption of dsp processor with minimal hardware overhead. We explained an incremental redesign of the TMS320c54x CPU to include the register-program table. We showed that pre configured and dynamic configured execution helps in achieving higher benefits for both execution time size and power with different tables. The register-program module saves the power by eliminates the instruction fetch stages and changes the state of program memory bus to high impedance state for entire lifetime of embedded application. We retain the memory interface speed since memory access times form the bottleneck in deep sub-micron designs. In our approach, the table implemented as a register file inside the CPU, the access delay is hidden. We used application programs such as benchmarks in our analysis. In our evaluations, we have taken into account the overhead due to register-program table. We observe that for execution time reduction and power reduction, larger tables give higher benefits. We showed that execution time reduction improvement of over 31% on an optimized code can be achieved and about 82% instruction fetch energy can be reduced. Further it is proposed to incorporate the existing methodology in on to an real time applications where the DSP processor plays an vital role in the over all operation and performance of the system. Also, where the power is an very major criteria and device is an battery operated and need to work for long time over a specific task.

REFERENCES

[1] Advance RISC Machines Ltd., “An Introduction to Thumb,” March 1995.

[2] L. Benini, G. Micheli, E. Macii, D. Sciuto, and C. Silvano, "Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems,"in Proceedings of the 7th Great Lakes Symposium on VLSI, pp. 77– 82, Urbana-Champaigne, IL, March 1997.

[3] L. Benini, F. Menichelli, and M. Olivieri, "A Class of Code Compression Schemes for Reducing Power Consumption in Embedded Microprocessor Systems," IEEE Trans. Comput., vol. 53, no. 4, 2004, pp. 467–482.

[4] Calculation of TMS320LC54x Power Dissipation," 1997, Application report, Texas Instruments. *http://www-s.ti.com/sc/psheets/spra164/spra164.pdf.*

[5] A. Chandrakasan and R. Brodersen, "Low Power Digital CMOS Design," Kluwer, 1995.

[6] S. Debray, W. Evan, R. Muth, and B. de Sutter, "compiler Techniques for Code Compression," ACM Trans. Program. Lang. Syst., 2000, pp. 378–415.

[7] M. Hiraki, R.S. Bajwa, et al., "Stage-Skip Pipeline: A Low Power Processor Architecture Using a Decoded Instruction Buffer," in Proceedings of International Symposium on Low Power Electronics and Design, Monterey, California, pp. 353– 358, Monterey, CA, August 1996.

[8] G. Subash Chandar, M. Mehendale, and R. Govindarajan, "Area and Power Reduction of Embedded DSP Systems using Instruction Compression and Reconfigurable Encoding", *Journal of VLSI Signal Processing*, vol. 44, pp.245-267, ~2006.

[9] L.H. Lee, W. Moyer, and J. Arends, "Instruction Fetch Energy Reduction Using Loop Caches For Embedded Applications with Small Tight Loops," in Proceedings of International Symposium on Low Power Electronics and Design, pp. 267–269, San Diego, CA, August 1999.

[10] M. Mehendale, S.D. Sherlekar, and G. Venkatesh, Bextensions to Programmable DSP Architectures for Reduced Power Dissipation," in Proceedings of International Conference on VLSI Design, pp. 37–42, Chennai, India, January 1998.

[11] S. Manne, A. Klauser, and D. Grunwald, "Pipeline Gating: Speculation Control for Energy Reduction," in Proceedings of International Symposium on Computer Architecture, pp. 132–141, Barcelona, Spain, June 1998.

[12] J. Rabaey and M. Pedram, "Low Power Design Methodologies," Kluwer Academic. Publishers, 1996.

[13] Siemens Incorp, BTriCore Architecture Manual," 1997.

[14] C. Su, C. Tsui, and A. Despain, "Saving Power in the Control Path of Embedded Processors," IEEE Des. Test Comput., vol. 11, 1994, pp. 24–30

[15] M. Stan and W. Burleson, "Bus-Invert Coding for Low Power I/O," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 3, 1995, pp. 49–58.

[16] Texas Instruments, BTMS320C54x DSP CPU and Instruction Set Reference Guide," June 1998. *http://acomms.whoi.edu/micromodem/TIDocs/c54x%20Ref%20Vol2%20Mnemonic%20Instr.pdf*

[17] TMS320C54x, LC54x, VC54x Fixed-Point Digital Signal Processors http://focus.ti.com/lit/ds/symlink/tms320lc541.pdf

[18] C.-Y. Wang and K. Roy, "Control Unit Synthesis Targeting Low-Power Processors," in Proceedings of IEEE International Conference on Computer Design, pp. 454–459, Austin, TX, October 1995.

[19] S.J.E. Wilton and N.P. Jouppi, "CACTI: An Enhanced Cache Access and Cycle Time Model," IEEE J. Solid-State Circuits, vol. 31, no. 5, May 1996, pp. 677–688.

[20] E.Beigne , Alexander Valentian,et al.,"A 460 MHz at 397mV,2.6GHz at 1.3V,32 bit VLIW DSP Embedding Fmax Tracking," in IEEE journal of solid state circuits,vol 50,No. 1,January 2015.

[21] Neeraj Magotra,Jim Larmier.,"energy efficient Digital signal processing", in IEEE 2010.

[22] Mahmut E Sinangil,Masood Qazi,et al., " Design of Low voltage Digital building Blocks and ADC for energy efficient systems," in IEEE transaction on circuits and systems-II: Express Briefs,Vol 59,No. 9,September 2012.

[23] Kyong Ho Lee, Naveen Varma.," A low power processor with configurable embedded machine-Learning Accelerators for higher- order adaptive Analysis of Medical-sensor signals.," in IEEE journal of solid state circuits,vol 48,No. 7,July 2013.

[24] Shen yu Peng, Lee et al.," Real time instruction cycle based dynamic voltage scaling(iDVS) power management for low – power digital signal processors(DSP) with 53% energy saving", in IEEE Asian solid-state circuits conference, November 12-14,2012 in Japan.

[25] Anadaroop Ghosh, Paul et al., " Improving energy efficiency in FPGA through Judicious Mapping of embedded memory blocks", in IEEE transaction on very large scale integration systems,vol. 22,,No. 6,june 2014.

[26] Yifan Bo, Renfeng Dou, Zeng."Hardware efficient variable length FFT processor for Low power applications".

[27] Dinko oletic, vedran. bilas,"Application of MSP430 and DSP C5000 in mobile patient and Environmental monitoring", in proceeding of the 6th European Embedded Design in education and Research,2014.

[28] Alessandro Dallai, Stefanoricci.," Real time bilateral filtering of ultra sound images through highly optimized DSP implementation", in IEEE proceeding of the European design in education and Research,2014.