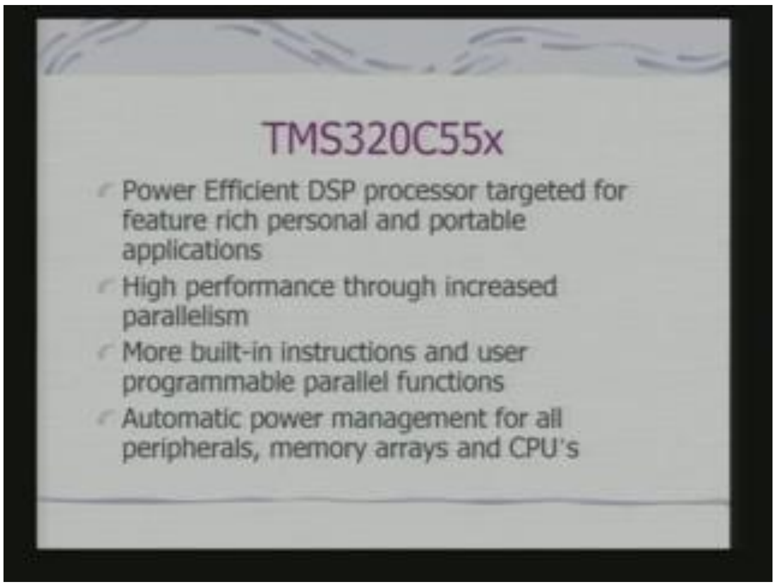
Embedded Systems
Dr. Santanu Chaudhury
Department of Electrical Engineering
Indian Institute of Technology, Delhi

Lecture - 09
More on DSP Processors

In the last class, we had discussed features of DSP processors. And we had looked at one of the DSP processors, from TI. That is C54X and that family. Today, we shall look at some more DSP processors and there architectures. The first processor that we are looking at, it is again from TI. And this is typically 55X architecture.

(Refer Slide Time: 01:43)

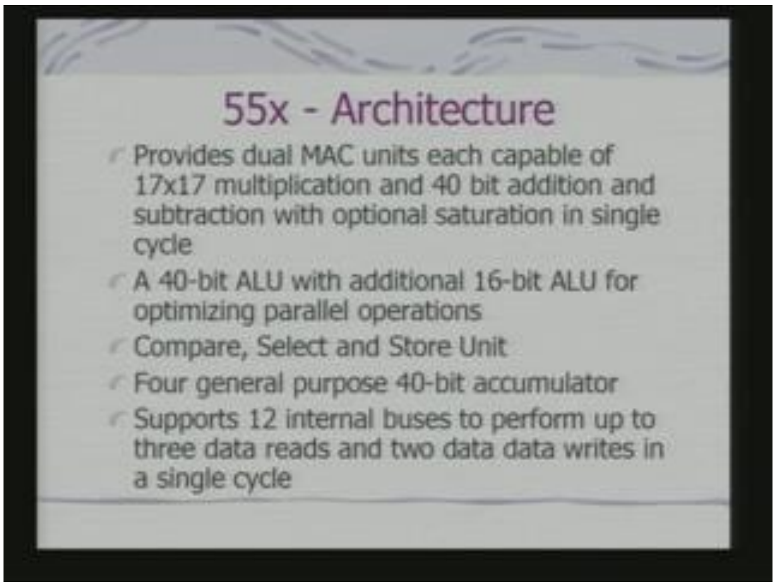


TMS320C55x

- Power Efficient DSP processor targeted for feature rich personal and portable applications
- High performance through increased parallelism
- More built-in instructions and user programmable parallel functions
- Automatic power management for all peripherals, memory arrays and CPU's

This is the power efficient DSP processor, which is really targeted for personal and portable applications. Like your cellular phone. And here, the basic different with earlier architecture is, to improve the efficiency with parallelism, in instruction execution. And another feature is automatic power management mode, which is in much more elaborate fashion, than what we had found earlier.

(Refer Slide Time: 02:24)



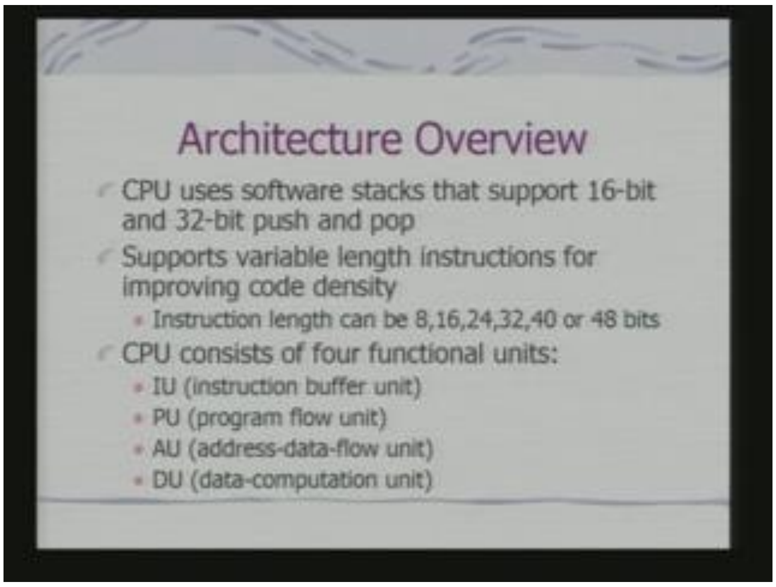
55x - Architecture

- Provides dual MAC units each capable of 17x17 multiplication and 40 bit addition and subtraction with optional saturation in single cycle
- A 40-bit ALU with additional 16-bit ALU for optimizing parallel operations
- Compare, Select and Store Unit
- Four general purpose 40-bit accumulator
- Supports 12 internal buses to perform up to three data reads and two data data writes in a single cycle

Architecturally you will find here that, the MAC unit. There are now two MAC units, that is it has be duplicated. And it has got the same 40-bit ALU. But, additionally there is an 16-bit ALU for optimizing parallel operations. It is supports 12 internal buses to perform up to three data reads; and two data writes in a single cycle. And you can understand these architectural modification has been done, to increase data throughput.

And consequently the output, that is the result. That is being generated, the rate at each the result can be generated that also can be increased.

(Refer Slide Time: 03:13)

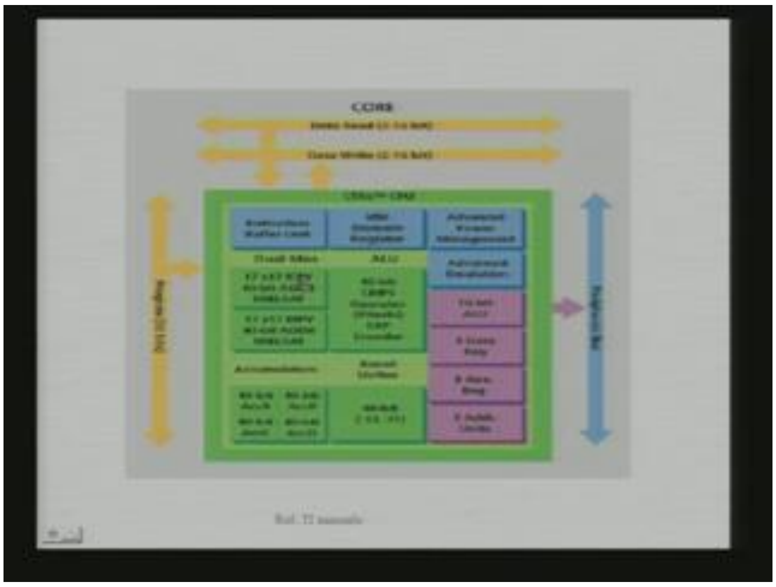


Architecture Overview

- CPU uses software stacks that support 16-bit and 32-bit push and pop
- Supports variable length instructions for improving code density
  - Instruction length can be 8,16,24,32,40 or 48 bits
- CPU consists of four functional units:
  - IU (instruction buffer unit)
  - PU (program flow unit)
  - AU (address-data-flow unit)
  - DU (data-computation unit)

In terms of the architecture overview, you will find that these processors supports variable length instructions. The advantage is that, you can have better code density. It users software stack, and not a hardware stack. So that, you can do a 32-bit push and pop. This CPU, the basic core processing unit, is divided into four sub units Instruction buffer units, program flow unit, address data flow unit and data computational unit. These unit facilitates multistage pipeline implementation, which again increases throughput.

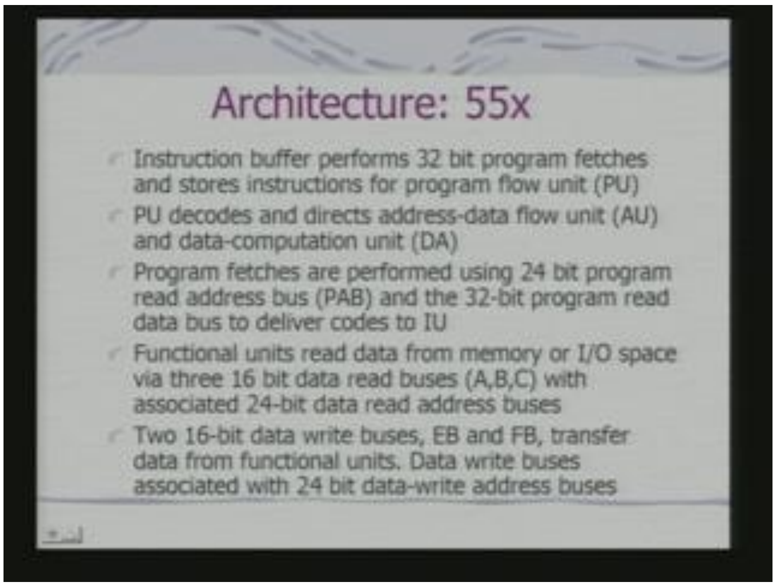
(Refer Slide Time: 04:14)



This is the basic diagram of the CPU core. We have not shown the basic components, but primarily the computational elements here. So, you will find that, it has got two 17 cross 17 MAC units. That is multiplier, adder as well as rounding and saturation hardware. It has got a 40-bit compare and select unit, then it has got four accumulators. So, obviously, I can accumulate more results. Also you have got a barrel shifter, it is a 40-bit barrel shifter.

Along with it, you will find you have got a 16-bit ALU apart from the regular ALU. It is got the 16-bit ALU, which is primarily used for address calculations. There are data and auxiliary registers. And you will find a block, which is dedicated for the power management. These are the buses, data read data write buses. This is a program bus, which is separated out. And this is a peripheral bus, for connecting it to peripherals.

(Refer Slide Time: 05:31)



Architecture: 55x

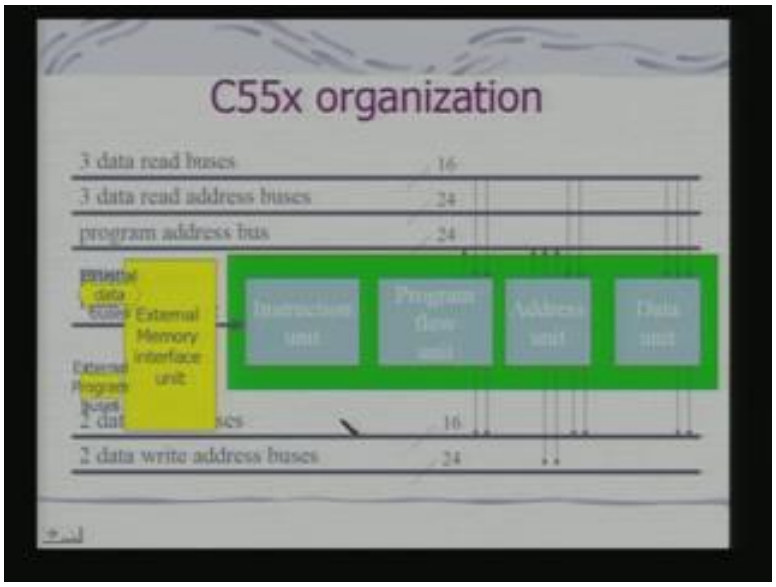
- Instruction buffer performs 32 bit program fetches and stores instructions for program flow unit (PU)
- PU decodes and directs address-data flow unit (AU) and data-computation unit (DA)
- Program fetches are performed using 24 bit program read address bus (PAB) and the 32-bit program read data bus to deliver codes to IU
- Functional units read data from memory or I/O space via three 16 bit data read buses (A,B,C) with associated 24-bit data read address buses
- Two 16-bit data write buses, EB and FB, transfer data from functional units. Data write buses associated with 24 bit data-write address buses

So, instruction buffer performs, what we call 32-bit program fetches and stores instructions for program flow unit. So, obviously it is responsible for implementation of what we call, pre-fetch in a pipeline stage. The program flow unit, decode and directs address data flow unit. And data computation unit, which comes subsequent to program data flow unit. In fact, program fetches are performed using 24-bit program read address bus.

So, address is 24-bit and what you get the words that is your instructions in chunks of 32-bits and functional units, read data from memory or IO space. These are functional units means, these are basically your either MAC units and ALU. The read data from memory or IO space via, three 16-bit data read buses. Each of which is associated with the 24-bit data read address buses. That means, I can do a simultaneous data read on this buses.

There are two 16-bit data write buses. So, these buses are exclusively mean for writing operations. So, all these operations can be actually overlapped in time. That is program read, data read and data write and that too on multiple data read and write buses.

(Refer Slide Time: 07:17)

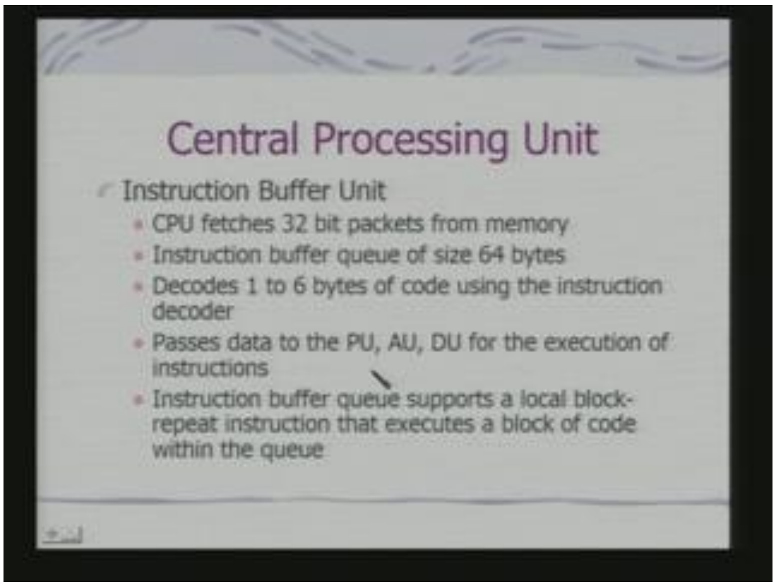

C55x organization

So, the organization is something like this. You have got an instruction unit, program flow unit, address unit, data unit. So, when you read an instruction, this instruction word which is a 32-bit. That means, instructions are fetched in packets of 32-bits becomes to instruction units. And the address is provided on the program address bus. Next, you will be reading the data, depending on the instruction being executed. There are three data read buses, each of which 16-bit with; and these are the data read address buses.

In fact, you will find various combinations of data reads and usage of buses, for that purpose. So, when you are using a single operand read, you will be using what is called a D bus inside the CPU. Dual operand read, two buses C and D buses. And dual multiply coefficients, that is you are trying to do two multiplications, simultaneously using the same coefficient. Then, the data that is a common coefficient will come along the B bus. You will be writing data via the write buses, and write address buses.

And these buses, what I have shown is, primarily the internal buses. And these buses will be interfaced to the external world. That is your external memory, by a special unit, which is called external memory interface unit.

(Refer Slide Time: 09:04)



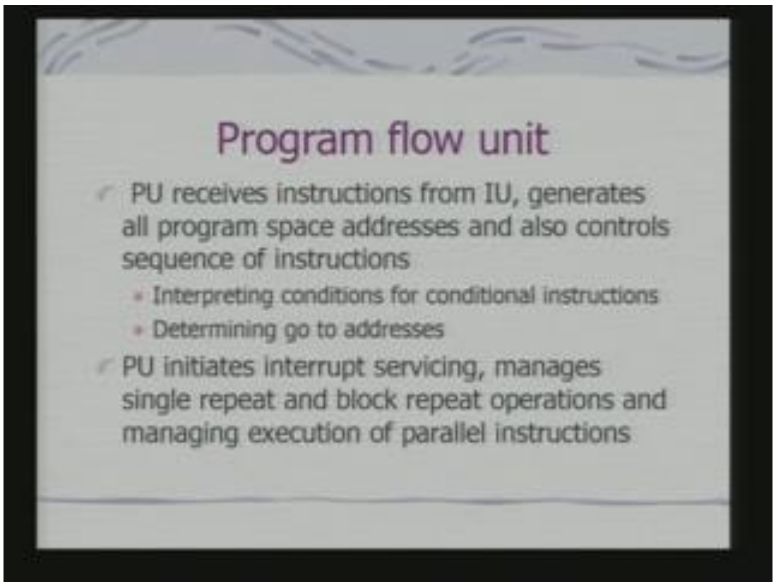
Central Processing Unit

- Instruction Buffer Unit
  - CPU fetches 32 bit packets from memory
  - Instruction buffer queue of size 64 bytes
  - Decodes 1 to 6 bytes of code using the instruction decoder
  - Passes data to the PU, AU, DU for the execution of instructions
  - Instruction buffer queue supports a local block-repeat instruction that executes a block of code within the queue

So, as I have already stated, that instructions buffer unit fetches 32-bit packets from memory. Instruction buffer queue size, is typically you have 64 bytes. That means that, many instruction bytes can be buffered and stored. It decodes 1 to 6 bytes of code using the instruction decoder. Once you decode, it passes the data to PU, AU and DU, for execution of instructions depending on the instructions, that has been actually fetched.

Instruction buffer queue, has an very important role to play is that of the implementation of a block repeat instruction. That means, when I am doing a block repeat that means, I am repeating a sequence of instructions. And where is this instructions stored, they can be stored in the instruction buffer unit. So, what the processor does, it executes as sequence of instructions, which is stored in the buffer.

(Refer Slide Time: 10:09)



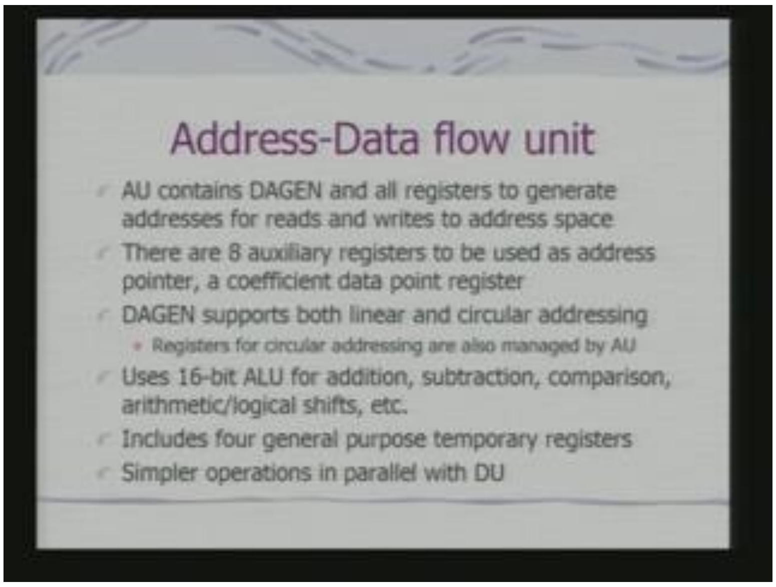
Program flow unit

- PU receives instructions from IU, generates all program space addresses and also controls sequence of instructions
  - Interpreting conditions for conditional instructions
  - Determining go to addresses
- PU initiates interrupt servicing, manages single repeat and block repeat operations and managing execution of parallel instructions

The program flow unit receives instructions from IU. Generates all program space addresses and also controls, the sequence of instructions. That is, it interprets the conditions, for conditional instructions. Because, here also you will find as in ARM, there are conditional instructions increase the code density and to have an efficient utilization of a pipeline and it also determines the go to addresses.

It initiate interrupts servicing, manages single repeat and block repeat operations. And managers execution of parallel instructions. In fact, this DSP has a facility to have parallel instruction, we shall come to that point, when you look at the instruction set. So, what you find, the program flow unit actually controls execution of the instruction. So, if I take the simple example of a kind of a block repeat, that block repeat management is done by PU but, the instructions are in the buffer. So, these units manages the exact control flow of the program, that is being executed.

(Refer Slide Time: 11:26)



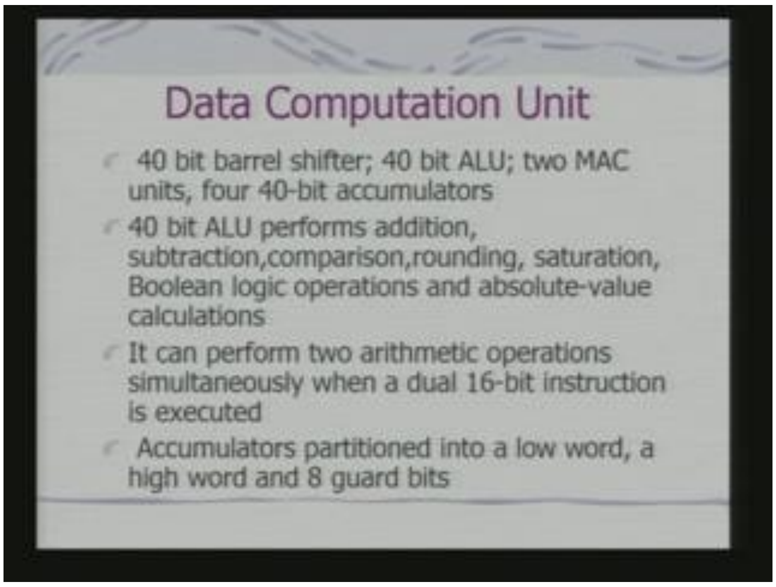
Address-Data flow unit

- AU contains DAGEN and all registers to generate addresses for reads and writes to address space
- There are 8 auxiliary registers to be used as address pointer, a coefficient data point register
- DAGEN supports both linear and circular addressing
  - Registers for circular addressing are also managed by AU
- Uses 16-bit ALU for addition, subtraction, comparison, arithmetic/logical shifts, etc.
- Includes four general purpose temporary registers
- Simpler operations in parallel with DU

Address data flow unit, which is actually you contains a logic for generation of addresses. It uses a set of registers, because actually you will find that, since you are having multiple data buses. You can actually get multiple operands. And you need to generate addresses, for this multiple operands. So, that is why address data flow unit, becomes a very important component for a DSP architecture. It uses a separate 16-bit ALU.

And these ALU is different from, that ALU’s which is being used for actual computation. And it includes four general purpose temporary registers. And simpler operations, can be executed in parallel with that of DU. That means, you can realize, that since there is a 16-bit ALU here. I can execute an instruction on the 16-bit ALU in parallel; with actually an instruction being executed on the data unit.

(Refer Slide Time: 12:27)

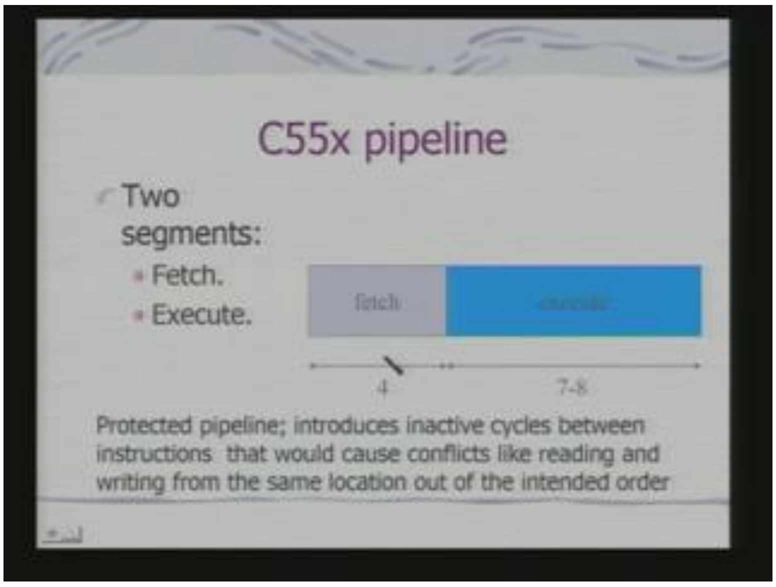


Data Computation Unit

- 40 bit barrel shifter; 40 bit ALU; two MAC units, four 40-bit accumulators
- 40 bit ALU performs addition, subtraction,comparison,rounding, saturation, Boolean logic operations and absolute-value calculations
- It can perform two arithmetic operations simultaneously when a dual 16-bit instruction is executed
- Accumulators partitioned into a low word, a high word and 8 guard bits

So, data unit basically consist of the 40-bit ALU, with the 40-bit barrel shifter. That I had already shown to MAC units, as well as four 40-bit accumulators. And you can perform two arithmetic operations simultaneously. When I do a 16-bit instruction is executed. In fact, you can understand this pretty well, that since you have got two MAC units. Obviously, I can execute two multiplication operations in parallel. Also with those multiplication operations, you can execute an arithmetic instruction on the 16-bit ALU, which is part of address data flow unit.

(Refer Slide Time: 13:13)



C55x pipeline

- Two segments:
  - Fetch.
  - Execute.

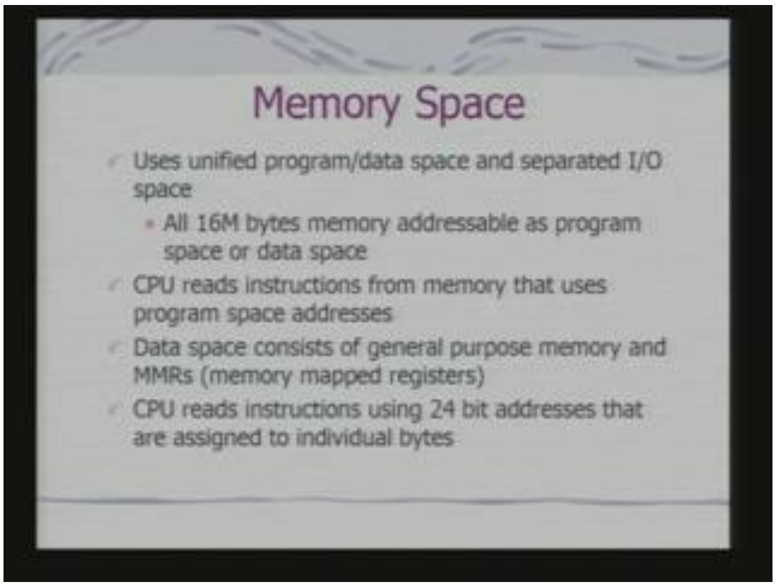
Protected pipeline; introduces inactive cycles between instructions that would cause conflicts like reading and writing from the same location out of the intended order

This pipeline, since we have got so many units. Obviously, it facilitates implementation of the multistage pipeline. In fact, we shall not go into the details of this stages but, we can say that the basic pipeline is divided into two phases, fetch and execute. And in fact, these 4 and 7 to 8, this numbers indicate number of sub-stages in each of this pipeline. So, it is a pretty extensive pipeline and the whole purpose is to increase the throughput.

That means, when you have the pipeline full and we are not having any problems related to pipeline flush, the instruction execution can be pretty fast. The other feature of this pipeline is, it is what is called a protected pipeline. It introduces in active cycles between instructions, that would cause conflicts. Like reading and writing from the same location, out of the intended order. So, that means, when there are two instructions in the pipeline, ready to be executed.

And there accessing the same location. And maybe one operand of an instruction, is dependent upon computation done by the previous instruction. Then, obviously, I cannot do a operand fetch, till my instruction is executed completely. So, in that case a pipeline stage has to idle so, I need to have inactive cycles. So that, is automatically introduced for efficient pipeline management.

(Refer Slide Time: 15:03)



Memory Space

- Uses unified program/data space and separated I/O space
  - All 16M bytes memory addressable as program space or data space
- CPU reads instructions from memory that uses program space addresses
- Data space consists of general purpose memory and MMRs (memory mapped registers)
- CPU reads instructions using 24 bit addresses that are assigned to individual bytes

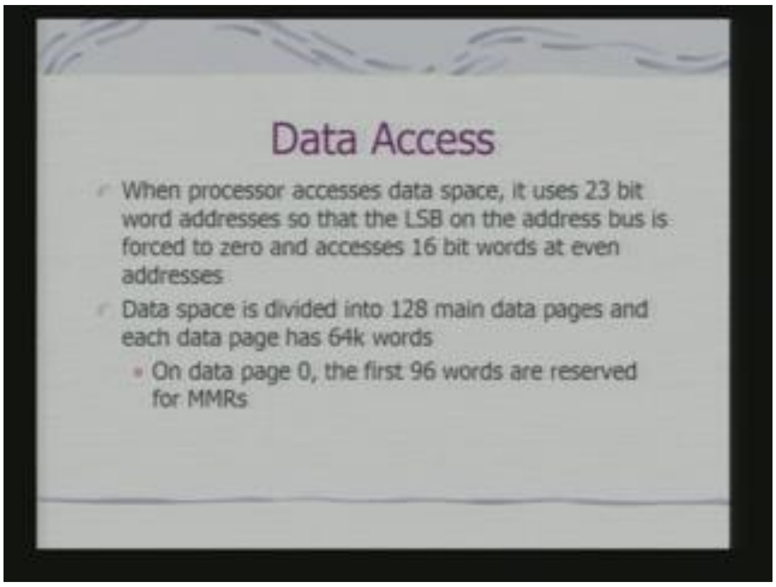
The memory spaces interesting, in case of see 55X although it is got a separate data buses, program buses. That is address, as well as program data bus. It can consider it is memory as a unified program data space, and separated IO space. In fact, it has got separate IO address space, separate program and data space. So, in fact, that all 16 megabytes of memories addressable, as program space or data space.

So, it is a kind of a modified Harvard architecture. It is not classical Harvard architecture such that, your address space and data space is absolutely distinct. And it can even have different sized words. So, here the whole memory space can be address together as a programs space, as well as data space. So, it is a modified Harvard. CPU reads instructions from memory, that uses program space addresses. And data space consist of, what we call a general purpose memory and MMRs, what does MMR?

MMR is a Memory Mapped Registers. That means, we are not talking about a separate set of registers, for doing data processing operations. We are referring to a set of memory locations. And these are one chip memory locations, which you are considering as memory mapped registers. In fact, you will find here similarity, with the concept of PIC registers. The PIC registers was part of the memory itself. The register bank itself is a RAM for PIC.

So, here there is a register memory mapped registers, as well as general registers. So, CPU reads instruction, using what we call to 24-bit addresses, you are found that all address passes or a 24-bits. So, addressing capability is 2 to the power 24. So, CPU read instruction using 24-bit addresses that, are assigned to individual bytes. So, therefore, this is again a byte addressable processor.

(Refer Slide Time: 17:13)

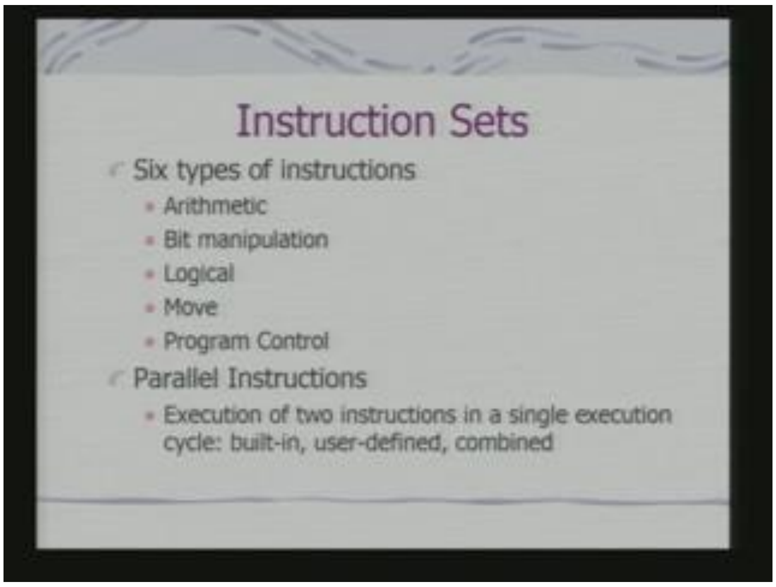


Data Access

- When processor accesses data space, it uses 23 bit word addresses so that the LSB on the address bus is forced to zero and accesses 16 bit words at even addresses
- Data space is divided into 128 main data pages and each data page has 64k words
  - On data page 0, the first 96 words are reserved for MMRs

Data accesses are interesting, because it accesses 16-bits at one word. And so effective address is 23-bits. And the whole data space is divided into 128 data pages; and each data page has got 64K words. And on data page 0, the first 96 words are reserved for a call memory mapped registers. Now, these whole memory, that we are talking about with the addressing capability of 2 to the power 24. It is not necessarily resident on the chip. Some part of it is on-chip and some part will be external.

(Refer Slide Time: 18:01)



Instruction Sets

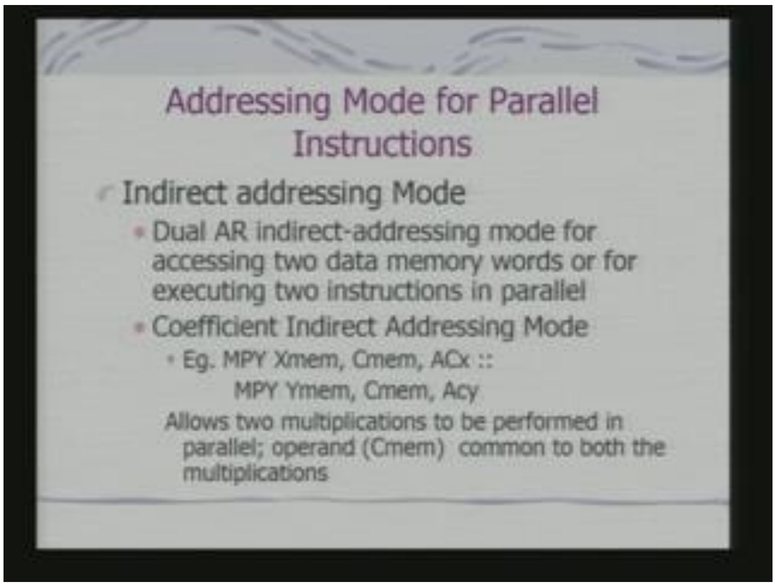
- Six types of instructions
  - Arithmetic
  - Bit manipulation
  - Logical
  - Move
  - Program Control
- Parallel Instructions
  - Execution of two instructions in a single execution cycle: built-in, user-defined, combined

Now, let us look at instructions sets. We shall not look at the basic instructions because, there is a distinct similarity of this instructions, with other processors that you have looked at. It has also got some special instruction which are targeted for signal processing applications. But, very interesting thing apart from all these is, the concept of parallel instructions. You have not come across this parallel instructions so far, in the architectures that you have discussed.

So, what are parallel instructions, here you can actually write a single instruction, which is part of your instruction set; which results in a execution of two instructions in a single execution cycle. Now, when it is part of the instruction set of the processors, we indicate such instructions as built in instructions. There is also provision for the user, to indicate such parallel instructions. So, they are user defined. And you can even combine system defined parallel instructions with additional instructions specified by the user and intended for parallel execution.

And that is what we refers to us combined form of parallel instructions. So, this is the features, which is unique to C55 among the processors that we have so far discussed. Obviously, you can understand the whole motivation is to make use of additional hardware. Increase the rate at which your data stream can be processed, input data stream can be processed. In the addressing mode, we are looking at in particular, the addressing mode which is used for this parallel instructions.

(Refer Slide Time: 19:53)



Addressing Mode for Parallel Instructions

- Indirect addressing Mode
  - Dual AR indirect-addressing mode for accessing two data memory words or for executing two instructions in parallel
  - Coefficient Indirect Addressing Mode
    - Eg. MPY Xmem, Cmem, ACx ::
      MPY Ymem, Cmem, Acy

    Allows two multiplications to be performed in parallel; operand (Cmem) common to both the multiplications

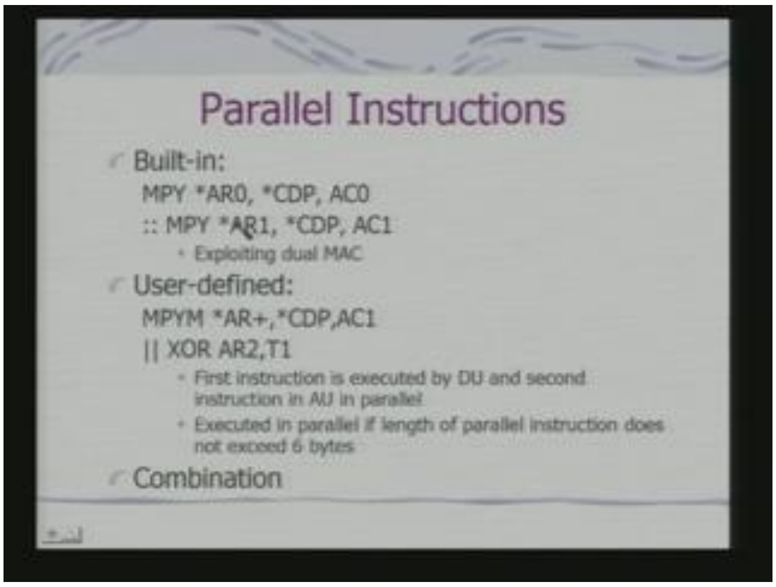
In fact, it uses indirect addressing. In fact, the registers which are available, for address purpose. They are address store purpose, there use in many cases for this address generations. So, you have got indirect addressing, where this AR indirect addressing typically used, when you are accessing two memory words. This two memory words will be required, for general instructions as well as when you are executing two instructions and parallel.

Because, if you are having the operand, so operands are in memory. So, you need to access this operands, two operand for two instructions. So, used typically and indirect addressing mode. And this indirect addressing mode is dual indirect addressing modes. There is other general addressing modes also available with this processor, which are commonly available. But, this dual indirect addressing mode is another distinct feature of this processor.

Other interesting thing is, coefficient indirect addressing mode. Here, this instruction is actually part of this instruction set of the C55. You can understand, this is a dual multiply instruction. And these is a symbol in the assembly language of C55, which indicates the built in parallel instruction. Here, this data that is X memory is actually being multiplied by a coefficient, which is stored in this Cmem. And the result is going to accumulate X, this again being multiplied and the result is going to accumulate Y.

So, two multiplication to be performed in parallel. And this operand, you will find this one is common to more the multiplications. So, you can realize the utility of having three buses. Since, you have got three buses and this one this data is common, between the two operations. So, using the three buses, I can actually execute two multiplications in parallel.

(Refer Slide Time: 22:02)



Parallel Instructions

- Built-in:
```
MPY *AR0, *CDP, AC0
:: MPY *AR1, *CDP, AC1
```
  - Exploiting dual MAC
- User-defined:
```
MPYM *AR+,*CDP,AC1
|| XOR AR2,T1
```
  - First instruction is executed by DU and second instruction in AU in parallel
  - Executed in parallel if length of parallel instruction does not exceed 6 bytes
- Combination

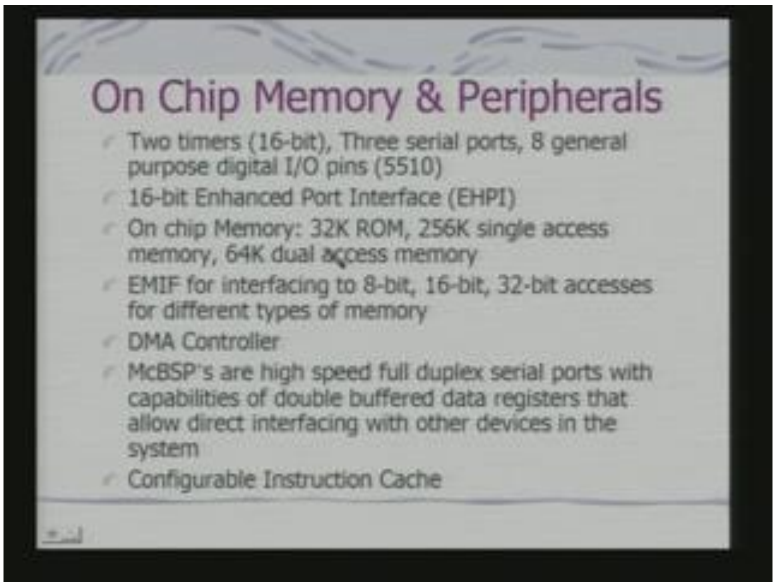
So, typically the instruction examples are something like this. Here, what we are doing, we are actually referring to this is star, is a standard, pointer, convention. This again a convention, which used in the assembly language of C55. We are pointing to memory locations. And you can find out that, these are through this register I am pointing to memory location, to distinct memory location. And this is a common memory location, which has got the coefficient value.

So, if you consider the same coefficient value, being used for two multiplication X and Y. I can use that and do to multiplication in parallel and exploiting dual MAC. The user can also defined this parallel instructions. If you look here, these an example where user has indicated in XOR operation. And this is a 16-bit operation. And these instructions, the first instruction is executed by the data unit. And second instruction is in ALU, which has got the 16-bit ALU.

Now, therefore, you will find, that user has got now flexibility of specifying parallel instruction. That is multiple instructions, being executed in parallel. Obviously, these instructions if there to be executed in parallel, certain rules and conditions have to be satisfied. One condition is that, that the parallel instruction does not exist 6 bytes. Why this 6 bytes restriction is coming in, from the instruction buffer because, it actually provide the 6 byte and decodes it.

So, both instructions should be decoded at the same time, and decodable in the instruction unit. So that, you know that they can be executed in parallel. Further, there should not be any hardware resource conflict. If there is a hardware resource conflict, then obviously, two instructions cannot be executed in parallel. And you have got the facility of combination, what is combination? So, this is an built in instruction so, with this built in instruction, you can even specify as a 16-bit operation. So, you can have got two multiplication plus a 16-bit operation, which can take place in parallel.

(Refer Slide Time: 24:30)



On Chip Memory & Peripherals

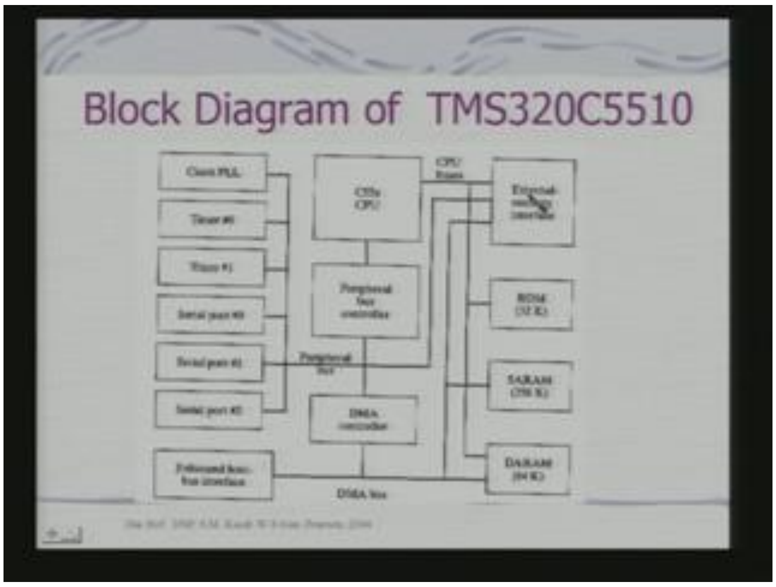
- Two timers (16-bit), Three serial ports, 8 general purpose digital I/O pins (5510)
- 16-bit Enhanced Port Interface (EHPI)
- On chip Memory: 32K ROM, 256K single access memory, 64K dual access memory
- EMIF for interfacing to 8-bit, 16-bit, 32-bit accesses for different types of memory
- DMA Controller
- McBSP's are high speed full duplex serial ports with capabilities of double buffered data registers that allow direct interfacing with other devices in the system
- Configurable Instruction Cache

Now, let us come to on chip memory and peripherals. I was referring to the memory access and data access. Now, which part is in on chip and which part can go outside. In fact, one ship memory, you will find this is 32K ROM, which is typically expected to be your program memory. You have got 256K single access memory and 64K dual access memory. So, these dual access and single access are required, because you are doing what, you may do a simultaneous read and write on to the memory.

So, in that case a dual access is important. You have got the timers, serial ports, 8 general purpose digital IO pins, which is typical to a particular version. It can change from chip to chip, because I am just present in the generic configuration. And interesting thing is that has got an enhance 16-bit enhance port interface; it has got EMIF. Now, this is for doing 16-bit interfacing with devices, IO devices. And this is for interfacing memory, external memory.

Apart from it you have got a DMA controller. And these are some special purpose serial ports, with buffering capabilities. It also provides for, what is call configurable instruction cache, which where... If you have the instruction the access is faster. And you can increase the data processing efficiency.

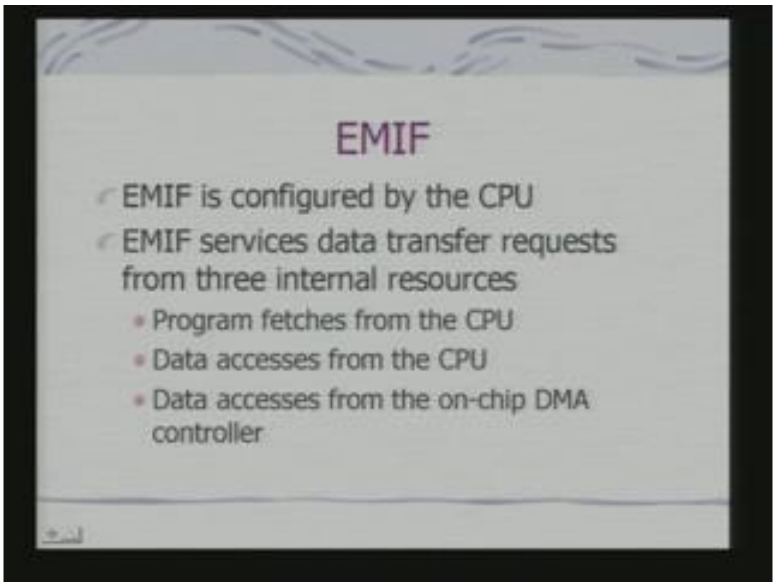
(Refer Slide Time: 26:19)



So, this is the basic block diagram other than we have so far discussed the CPU part of it. Apart from CPU, on the chip itself there are other peripherals. So, what you can see is, this is my DMA controller which connected by a peripheral bus controller. If you remember the first diagram, I have shown a peripheral bus. So, peripheral bus controller, controls the peripheral bus. And on the peripheral bus is actually all this, this timers, serial ports, all this sits on the peripheral bus. And this is your DMA controller.

And DMA controller has itself a DMA bus, which is this interface to the external world, through this enhance port bus interface. And these are basically memory blocks. This is the ROM, this is single access RAM, this is basically dual access RAM. And through this external memory interface block, you can connect to external memory. And CPU buses are therefore, connected to this external memory interface. If I need accessing external memory, as well as to the internal memory locations.

(Refer Slide Time: 27:40)



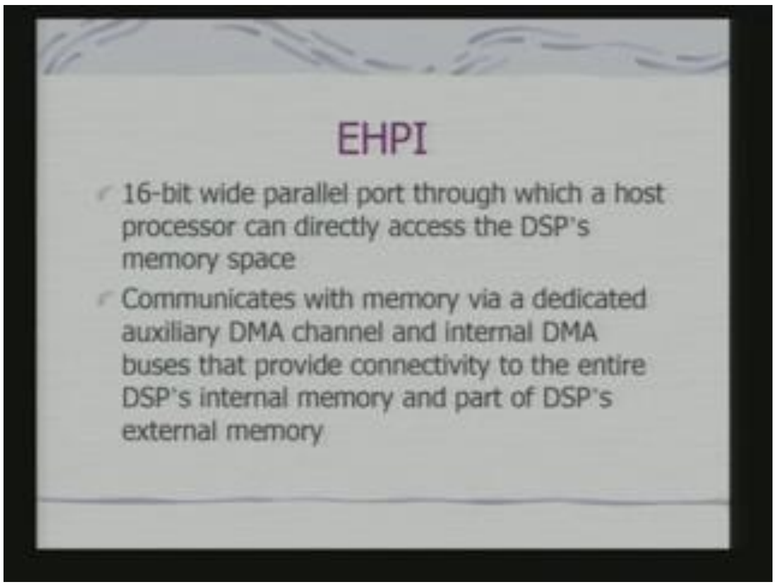
EMIF

- EMIF is configured by the CPU
- EMIF services data transfer requests from three internal resources
  - Program fetches from the CPU
  - Data accesses from the CPU
  - Data accesses from the on-chip DMA controller

Now, EMIF is actually the block, which decides on the interface issue. So, interfaces configure by the CPU and it is a hardware, which services data transfer request from three internal resources. That is program fetched from the CPU, data accesses from the CPU. And data accesses from the one chip DMA controller. And in fact, the whole idea here is, that there is a specialized hardware, which enables you the chip to interface to different external memory devices.

Because, this devices can have different timing constraints they can store, that means they are inherent storage, can be of different deadlines. It maybe on 8-bit memory, it may be a 16-bit memory. So, all this interface issues, are being managed by a separate controller. And that is exactly what is this EMIF. So, CPU is not really consigned with this management task of interfacing, with external memory devices.

(Refer Slide Time: 28:57)



EHPI

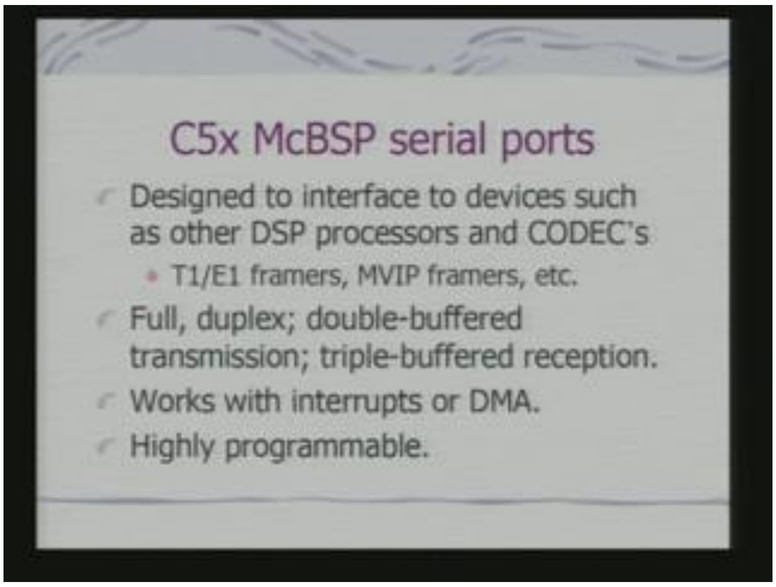
- 16-bit wide parallel port through which a host processor can directly access the DSP's memory space
- Communicates with memory via a dedicated auxiliary DMA channel and internal DMA buses that provide connectivity to the entire DSP's internal memory and part of DSP's external memory

Similarly, we have got EHPI, which is a 16-bit wide parallel port through, which a host processor can directly access DSP's memory space. Now, this is an interesting feature. Why, because you have can be a general purpose host, which can used DSP as a coprocessor. So, I need to provide a general purpose external host, some access to DSP. So, this EHPI provides that interface. So, this can be connected to an external host. In fact, it can be connected to other peripheral also if required.

Now, EHPI communicates with memory, that is it is provides communication with memory. If you remember the block diagram, the DMA controller was connected. To this via dedicated auxiliary DMA channel; and internal DMA buses. That provides connectivity to the entire DSP's internal memory and part of DSP's external memory. So, what is this significance of this, significance is that external host. If maybe a processor, it maybe CODEC's also, can transfer the data onto the DSP's memory for processing.

Even CPU can influence, what the code the DSP execute. It may even generate and write the code in the DSP's memory. The DSP's expected to execute depending on the task which is being given to the host for execution.

(Refer Slide Time: 30:35)

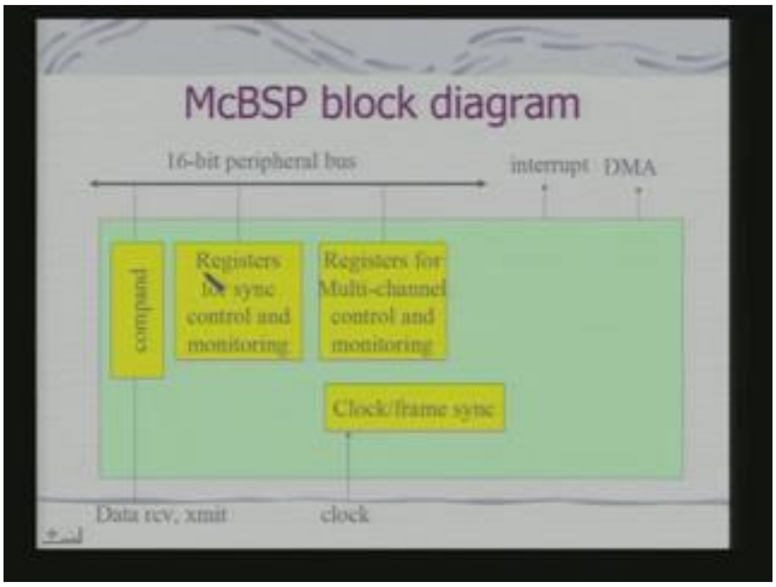


C5x McBSP serial ports

- Designed to interface to devices such as other DSP processors and CODEC's
  - T1/E1 framers, MVIP framers, etc.
- Full, duplex; double-buffered transmission; triple-buffered reception.
- Works with interrupts or DMA.
- Highly programmable.

These are serial ports. And this says, this serial ports are typically buffered serial ports. And these are special serial ports, you will find in many of the DSP's. Not only on these, but other DSP's as well designed to interface to devices, such as other DSP's processors. This is got a specific form, it is double buffered transmission and triple buffered reception. So, you can understand the buffering means, there can be the timing synchronization with external device, can be done by the DSP using the buffers. So, it can be full duplex, double buffered transmission, triple buffered reception. It works with interrupts or DMA and it is highly programmable.

(Refer Slide Time: 31:30)

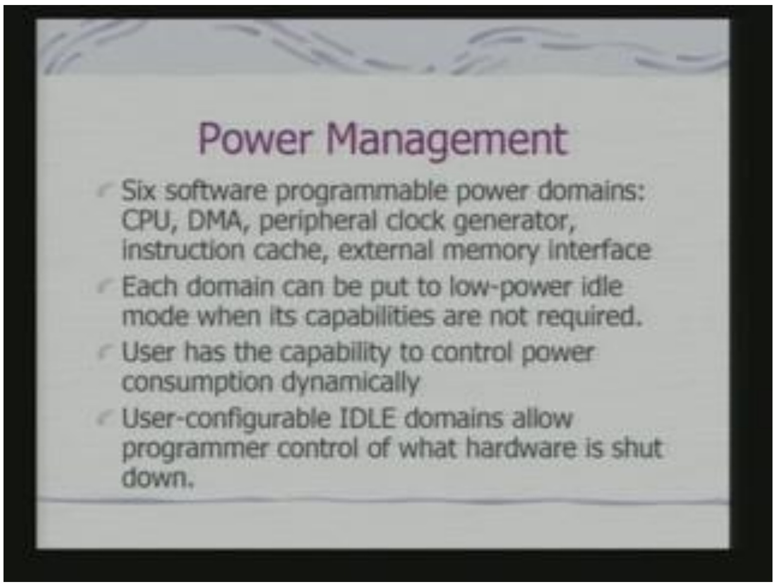


McBSP block diagram

16-bit peripheral bus
interrupt
DMA
compand
Registers for sync control and monitoring
Registers for Multi-channel control and monitoring
Clock/frame sync
Data rcv, xmit
clock

In fact, if you look at the basic block diagram if you see, that you can program it for register, for synchronization control, sync control. Sync, it is a serial transmission, so I can do a sync control and monitoring. Registers from multi-channel control and monitoring, and I can specify the clock and frame synchronization. Because, this is a clock signal, because of a serial synchronize transfer I need to have a clock. So, these are the lines for data receive or transmit, and it is buffered. So, it can also actually raise and interrupt request, or DMA request for a data transfer. So, these provides you with the serial interface with external devices, which can be even another DSP.

(Refer Slide Time: 32:18)



Power Management

- Six software programmable power domains: CPU, DMA, peripheral clock generator, instruction cache, external memory interface
- Each domain can be put to low-power idle mode when its capabilities are not required.
- User has the capability to control power consumption dynamically
- User-configurable IDLE domains allow programmer control of what hardware is shut down.

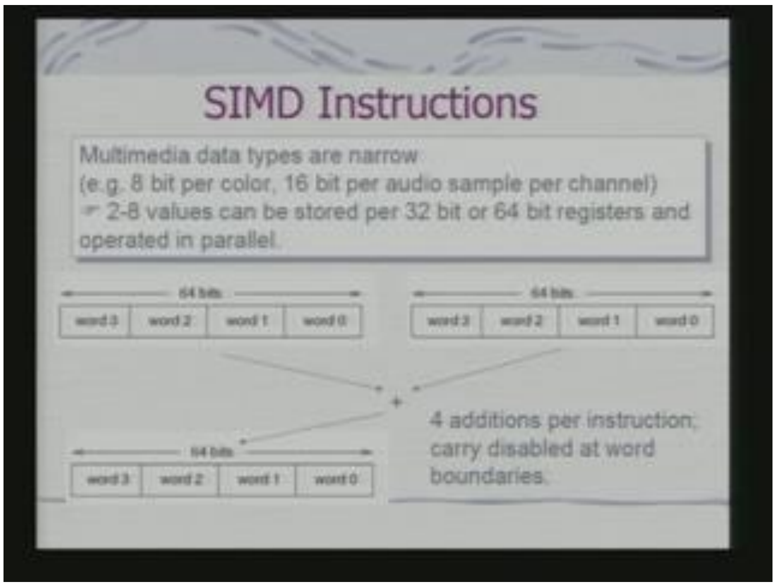
Next, we look at power management. In fact, we have remember that there is a dedicated power management block. Now, how does really, the power management comes into play. You have found C55 has got various functional blocks, it is not that for all task, all functional blocks are required. So, what has been done is the designers have got 6 software programmable power domains, on the processors. They are CPU, DMA, peripheral clock generator, instruction cache, external memory interface.

And each domain can be put to low power idle mode, when it is capabilities are not required. And this is completely user and user control that is, software control. User even defined ideal domains. So, with this ideal domain, it can put some parts in ideal mode; and reduce the power consumption depending on whether the task request this features or interfaces or not. Take for example, when I know in my software, I shall not be accessing the external memory. I can definitely put external memory interface in ideal state and safe power.

So, if you remember, I started with say in that C55 is targeted for a power efficient DSP architecture. And this is one of the manifestations of these, that facility to defined ideal domains. And can put the different one chip blocks to ideal mode, under software control. Now, if you compare C54 with 55, what we have found is, enhancement in the architecture primarily to increase the weight, at which data can be processed. But, still this is a fix point architecture.

And a very distinct thing that you have observed, there is ability to execute multiple instruction in parallel because of the additional hardware. Now, we shall look into some variants of this DSP architecture but, these feature has been explored in full.

(Refer Slide Time: 35:03)



SIMD Instructions

Multimedia data types are narrow
(e.g. 8 bit per color, 16 bit per audio sample per channel)
☞ 2-8 values can be stored per 32 bit or 64 bit registers and operated in parallel.

4 additions per instruction; carry disabled at word boundaries.

64 bits
word 3
word 2
word 1
word 0
64 bits
word 3
word 2
word 1
word 0
+
64 bits
word 3
word 2
word 1
word 0

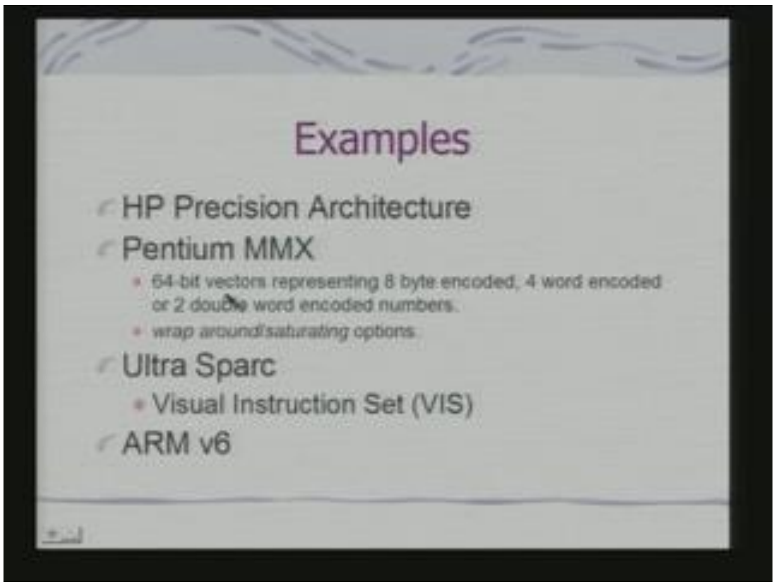
We have already come across a SIMD instructions. Now, SIMD instructions are basically meant for multimedia data types. In fact, the examples that I have started with say I need sensor. And from each side I can get a 8-bit data. And if I am doing any kind of image processing operation, I shall be operating on each and every pixel. So, I shall be operating on 256 and cross 256 pixels if my image size is 0 to 255 by 0 to 255 that means, the squared image.

So, that many operations are to be performed, and these same operations and repeated operations. Similar thing is true, even for audio signals. You are taking samples at regular intervals and similar operation will be perform on each and every sample. So, there is a motivation for what we call SIMD instructions. That is single instruction multiple data. In fact, we have seen this in the context of ARM version 6 architecture as well.

So, effectively what we are telling is, if I have really 32-bit or 64-bit registers in the processor. I can logically divide this processors into, say 4 words in this case I have done for a 65-bit. And I have done in a addition. And here the addition is word wise addition. And what we say that four additions per instruction; and carry is disabled at one boundaries. We can even have a saturation mode. And that effectively is doing what, reducing the number of instructions actually need to be store as well.

So, you can realize that code density also increases. So, in fact this sample instruction for this kind of multimedia processing is today common, not only for embedded processors. But, also for general purpose processor as well.

(Refer Slide Time: 37:18)



Examples

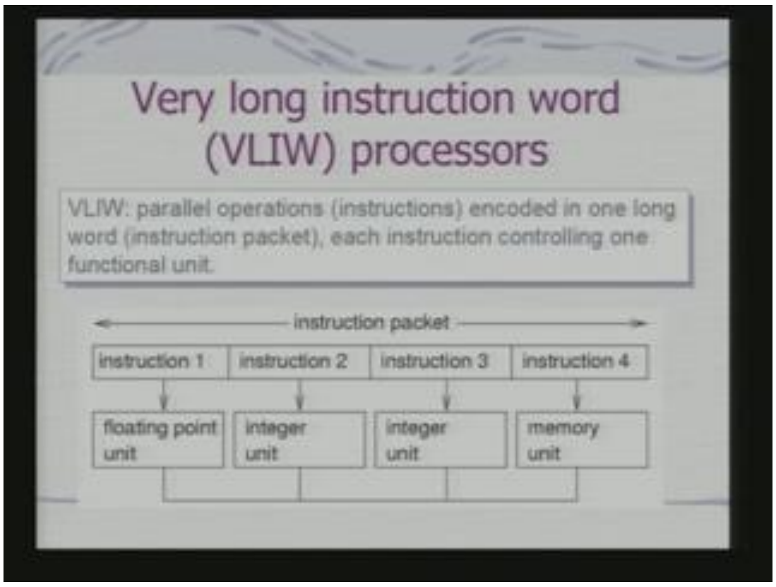
- HP Precision Architecture
- Pentium MMX
  - 64-bit vectors representing 8 byte encoded, 4 word encoded or 2 double word encoded numbers.
  - wrap around/saturating options
- Ultra Sparc
  - Visual Instruction Set (VIS)
- ARM v6

Here, we have given some examples. HP precision architecture is an old architecture, one of the earlier architecture which implemented this. This Pentium MMX, which is coming with all of your PC’s. They have got 64-bit vectors representing 8 byte encoded, or 4 word encoded or 2 double word encoded numbers. That means, you can have these kind of combinations are arithmetic instructions. You can have 8 byte encoded in 64-bit. So, if you have doing an add, in that case you have 8 additions taking place in one instruction cycle.

Similarly, if you have two double word, then many two double word additions in one instructions cycles. Ultra Sparc has got an instruction set, which is visual instruction set. In fact, there the instruction set, one of the features of this instruction set is this has been optimized, for processing of the video data. Because, you will find that today, if you are getting your cameras or even your cell phones, ability to capture video, encode video, that is compress video, stores and etcetera. Then, I need first hardware to do video compression.

In fact, the first such features this was, this is not for really embedded system, this was done on the work station level. But, they introduce special instruction sets, which had assigned instructions to facilitate video processing. And ARM version 6, actually that we had discussed in one of the previous classes. Next use of this SIMD instructions, for these kind of signal processing tasks, for embedded domain.

(Refer Slide Time: 39:03)



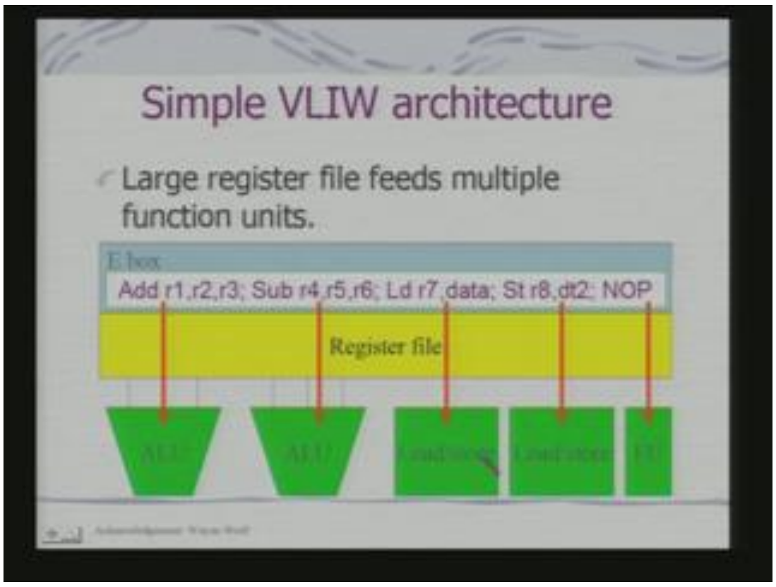
Very long instruction word (VLIW) processors

VLIW: parallel operations (instructions) encoded in one long word (instruction packet), each instruction controlling one functional unit.

Now, a next thing is what is called very long instruction word processors, or VLIW processors. Here, I have got parallel operations, encoded in one long word, each instruction controlling 1 functional unit. So, we say that, this is an instruction packet. This is an instruction 1, this is an instruction 2, this is instruction 3 and this is instruction

4; all part of a single instruction packet. And there activating different functional units, in parallel. And in fact, this is exactly the extension or enhancement of that feature, that we are found in C55. But, in C55 it was not an explicit VLIW instructions. But, when we have the VLIW processor, explicitly with the multiple units, I can have this kind of long instruction words. In which multiple instructions can be specified for execution in parallel.

(Refer Slide Time: 40:11)



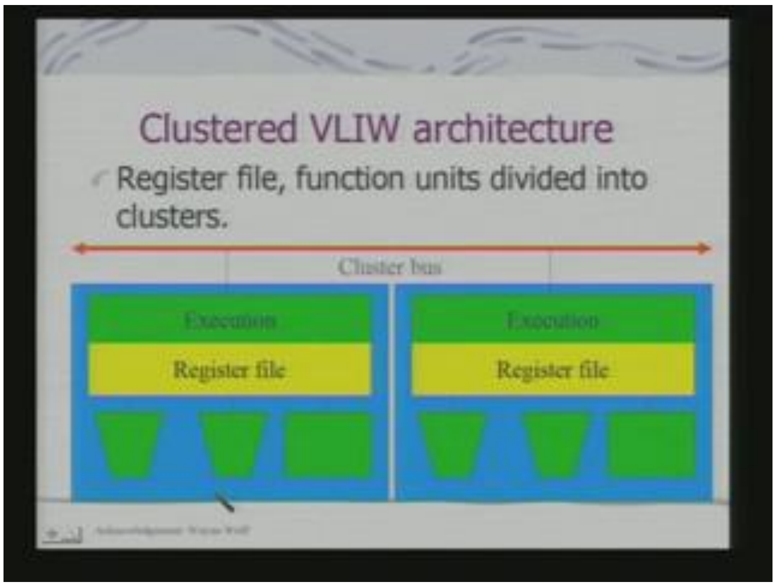
Simple VLIW architecture

- Large register file feeds multiple function units.

So, in this case what happens, if we consider this, then we need a large register file, which would feed multiple function units. So, if you take an example, let us say I have got this instructions. A very large instruction what architecture, in this case an instruction packet we will have op-codes for say addition. Register addition subtractions, load or store or NOP. The interesting feature is you obviously, have seen here just like I was talking about parallel instruction.

They should not be any hardware resource conflict, that means a registers that we have using or different. Therefore, these instructions, which are to be executed they should feed into the ALU, the data from the corresponding register file. So, effectively these instruction; and the data from this registers could going to here. And the result should be stored back similarly, true for subtraction. And interestingly here I have shown, even other than ALU or arithmetic units, we have got load store units. This load store units taking care of load and store equals.

(Refer Slide Time: 41:37)

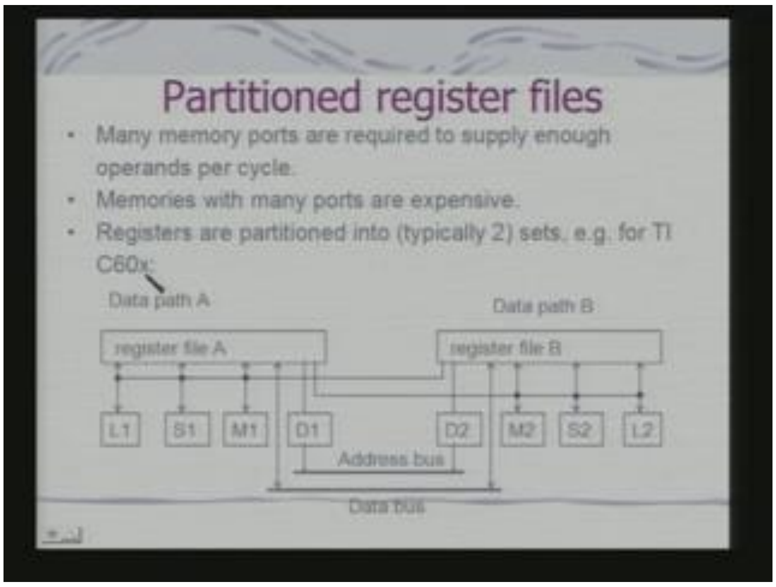


Clustered VLIW architecture

- Register file, function units divided into clusters.

Now, register the other option, the organizational option for VLIW architecture. Is that instead of having a simple register file, feeding into all the functional units. You can have register file and function units divided into what are called, cluster. And there is a common cluster bus. And from there, you feed into the execution units. And each execution unit is associated with it is, all dedicated set of register files, which is used for doing computation, using multiple functional units.

(Refer Slide Time: 42:19)



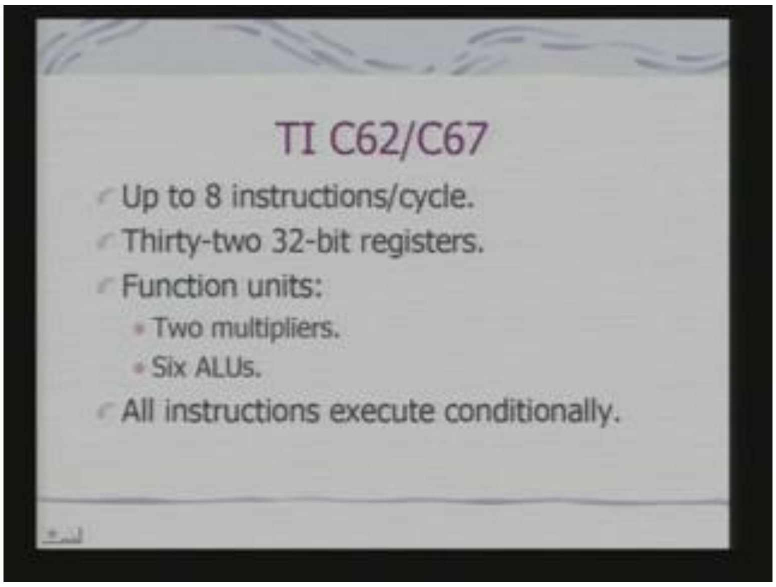
Partitioned register files

- Many memory ports are required to supply enough operands per cycle.
- Memories with many ports are expensive.
- Registers are partitioned into (typically 2) sets, e.g. for TI C60x:

A similar to that cluster concept, is this implementation which is in TI C60x family. We have look at C55x family now, we are looking at the 60x family, which is actually an explicit VLIW processor. The whole architecture is motivated on VLIW processor. And you will find that in this case obviously, the many memory ports are required to supply enough operands per cycle. So, what is being done here is, that whole data path, we have got its feeds into the two register bank set.

Register file A and register file B and their feed into this functional units. So, these are address buses, if there generating the address units and these are the data buses, which are feeding onto the register files. Because, if I am getting a data from the memory or one register file to another register file, I need to have a data bus connecting them together. And these are the functional units, which are connected directly to two different register files. So, this is the basic organization of 60x family of TI processors, which are VLIW digital signal processors.

(Refer Slide Time: 43:49)

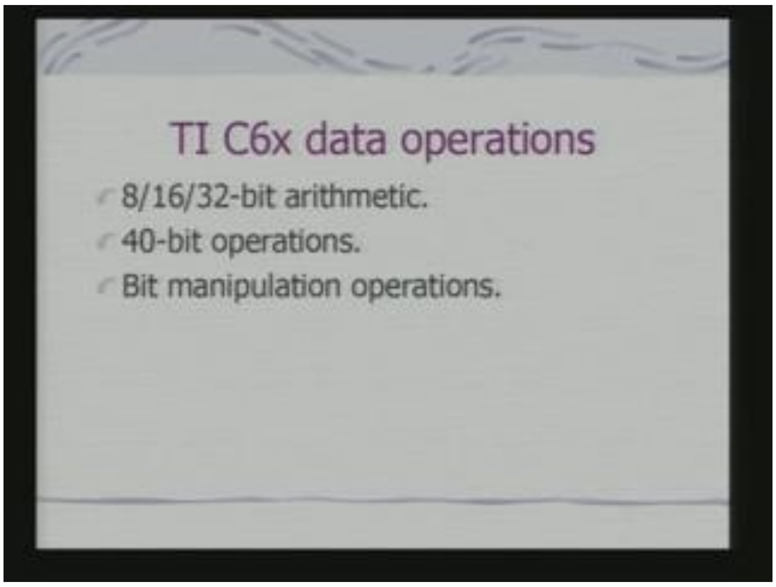


TI C62/C67

- Up to 8 instructions/cycle.
- Thirty-two 32-bit registers.
- Function units:
  - Two multipliers.
  - Six ALUs.
- All instructions execute conditionally.

So, we are looking at 62, 67 some examples of this processor. They can execute up to 8 instructions in parallel per cycle. Obviously, you will find this is definitely an enhancement over your C55. It has got 32-bit registers. And these are the functional units, that is two multipliers and six ALU's. And it all instructions execute conditionally. So obviously, if some instructions did not be executed, that maybe part of a long instruction word. And that function units is not used, so go executes what we call an in active cycle.

(Refer Slide Time: 44:40)



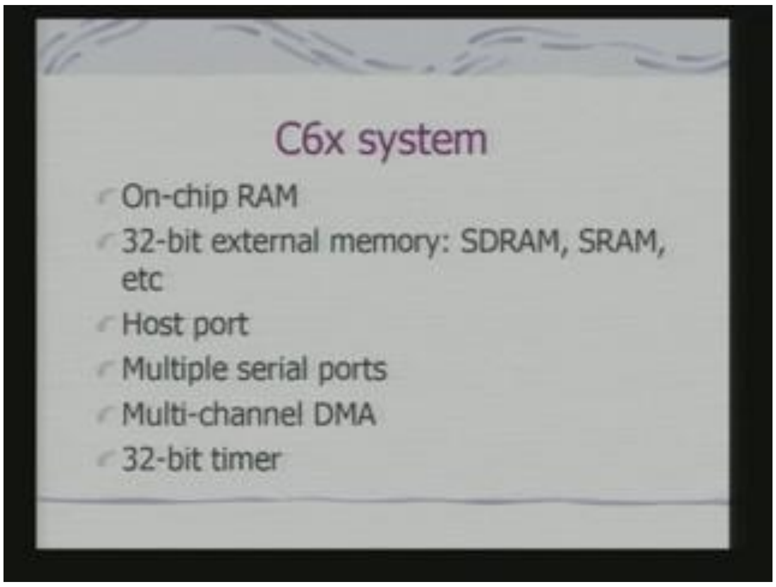
TI C6x data operations

- 8/16/32-bit arithmetic.
- 40-bit operations.
- Bit manipulation operations.

So, it is got 8, 16 and 32-bit arithmetic, as well as 40-bit operations, it also has got bit manipulation operation so, variety of operations which are available. So obviously, you can find that ability to process so many arithmetic operations and parallel. Increases it is capability to handling, huge volume of data. And that exactly the motivation, that when we are looking at this media data, particularly. It is not just a data coming from a ((Refer Time: 45:10)).

But, really media sensors, like image and speech audio. You really get high volume data, and new to do substantial amount of processing on this data, for doing compression and decompression, basic tasks. And therefore, these kind of processors are targeted for this applications.

(Refer Slide Time: 45:34)



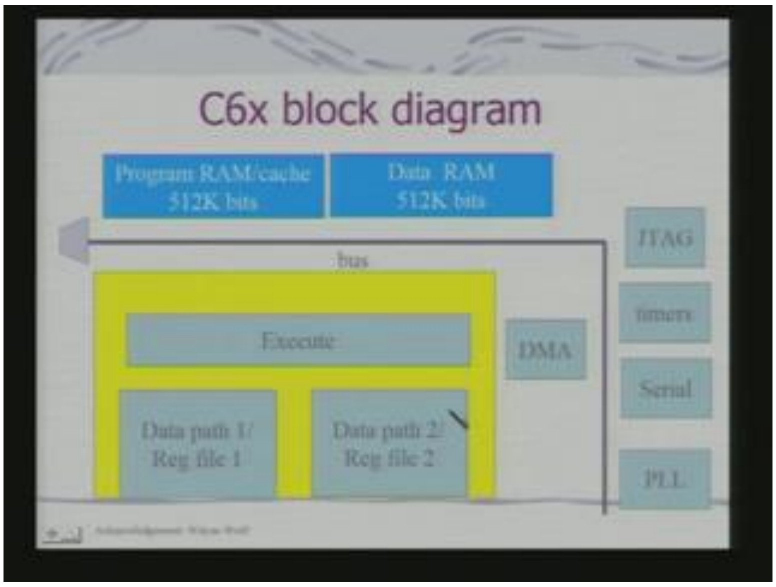
C6x system

- On-chip RAM
- 32-bit external memory: SDRAM, SRAM, etc
- Host port
- Multiple serial ports
- Multi-channel DMA
- 32-bit timer

In fact, just like your C55x family, C6x family has got on-chip RAM. It can have 32-bit external memory. It has also got a host port, multiple serial ports, multi-channel DMA 32-bit timer. These are standard peripheral block, which you expect on many of the other processors as well.

(Refer Slide Time: 45:51)

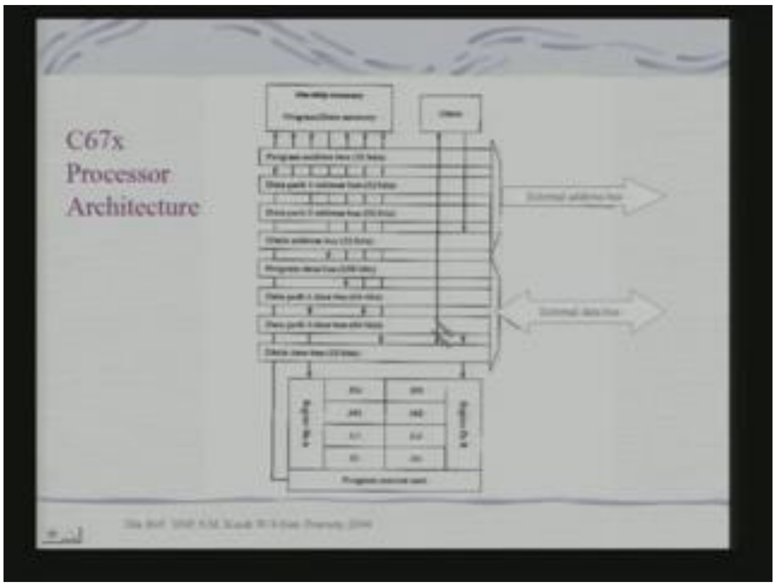
C6x block diagram



So, these is the basic idea, a what basic organization block diagram. So, it has got a program RAM or cache, data RAM or cache, the same bus. It has got a DMA controller.

And there are two data paths. Along this data path that have two register file banks, which feed into the execution units.

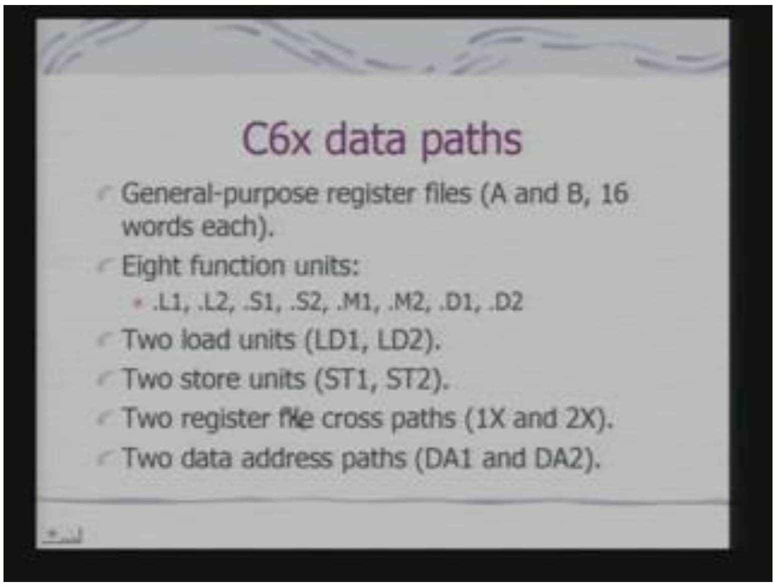
(Refer Slide Time: 46:16)



This is a 67X processor architecture. The reason why I have shown this diagram is, you will find that, this many internal buses have been provided. This is one chip memory and this is your DMA controller. This is just the same on, exact implementation of the previous generic architecture will background, that I have shown. And these are the basic functional units, D1, M1, L1 and X1. So, these are the basic functional units. It has got the register file A and register file B, feeding into this two functional units independently.

And the program control unit knows, how this functional units have to be used. And these are the data path. And interestingly you will find, the data is got the data paths, 32-bit is a DNI data bus. And these are 64-bit data buses, even program data buses are 256-bit. Obviously why, because it is a VLIW processor and it is instruction size would be large. That is why the program data bus, has to be 256-bits. And this is the very basic feature of any of the VLIW processors.

(Refer Slide Time: 47:38)

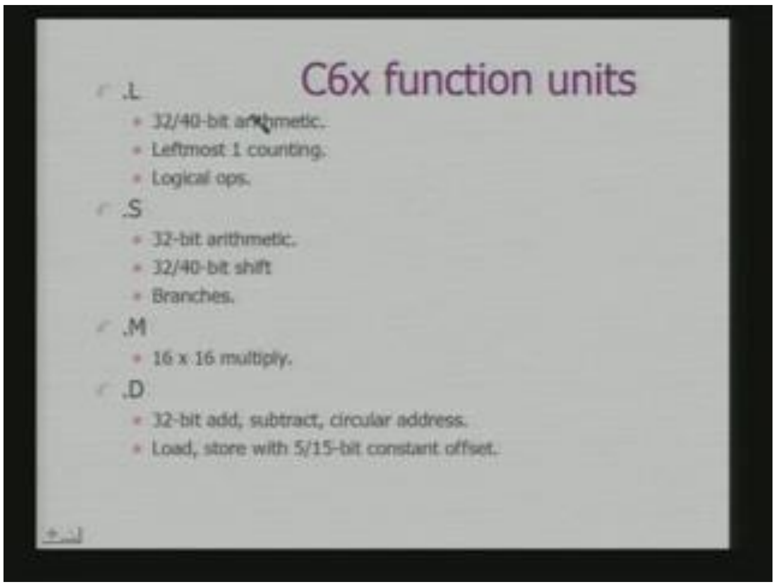


C6x data paths

- General-purpose register files (A and B, 16 words each).
- Eight function units:
  - .L1, .L2, .S1, .S2, .M1, .M2, .D1, .D2
- Two load units (LD1, LD2).
- Two store units (ST1, ST2).
- Two register file cross paths (1X and 2X).
- Two data address paths (DA1 and DA2).

So, as we have seen in the data path, that you have got general purpose register files. 8 function units, 2 load units, 2 store units, two register file cross paths and two data address paths. So, let us look at what are this functional units really doing.

(Refer Slide Time: 47:54)

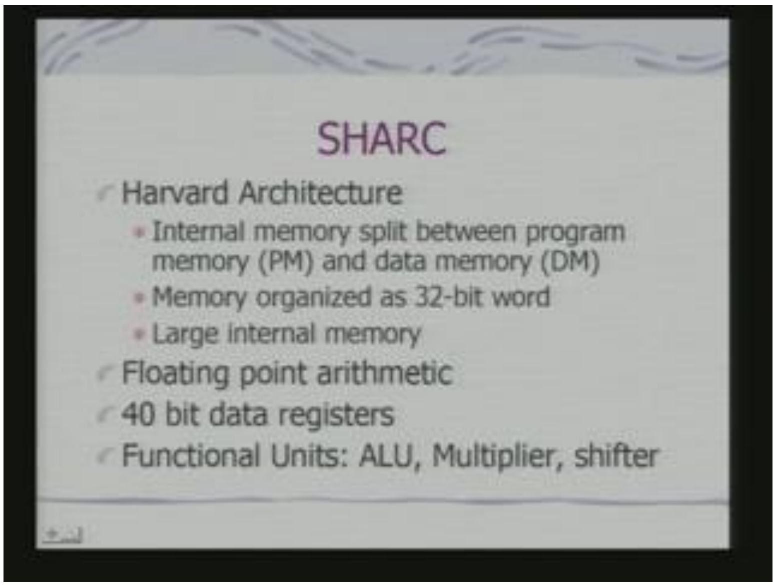


C6x function units

- .L
  - 32/40-bit arithmetic.
  - Leftmost 1 counting.
  - Logical ops.
- .S
  - 32-bit arithmetic.
  - 32/40-bit shift
  - Branches.
- .M
  - 16 x 16 multiply.
- .D
  - 32-bit add, subtract, circular address.
  - Load, store with 5/15-bit constant offset.

Now, among various tasks, and just representing some of the basic tasks. This L represents basically 32 or 40-bit arithmetic, it can be logical operation, it can count the left most one counting. S is also doing 32-bit arithmetic and 30 to 40-bit shift and it can do branches. So, this shifting is, what is distinguish S from L blocks, M is basically the multiply units. And D is the address, basically manages address, as well as load store. And so if you remember in the basic block diagram, on the organization diagram I showed, by the D block is connected to the address bus. So, it D do 32-bit add subtract circular address management etcetera.

(Refer Slide Time: 48:47)

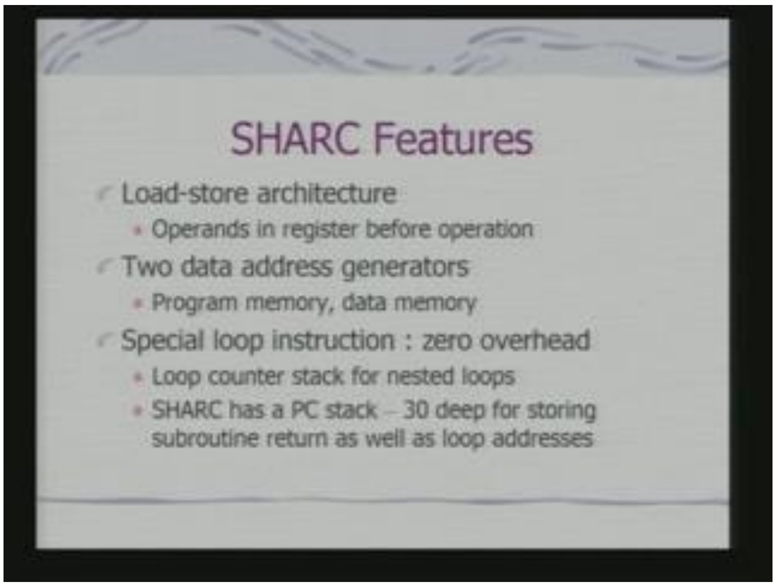


SHARC

- Harvard Architecture
  - Internal memory split between program memory (PM) and data memory (DM)
  - Memory organized as 32-bit word
  - Large internal memory
- Floating point arithmetic
- 40 bit data registers
- Functional Units: ALU, Multiplier, shifter

Now, similar to C6X, the SHARC processor SHARC is again another DSP, this is coming from different manufacturer, which also implements Harvard architecture. It implements the memory, divided into two blocks, PN and DN. But, in the most important feature of SHARC is a, it is a floating point DSP processor. So, it has got a floating point arithmetic unit inside; and it has got floating point instructions. The functional units are basically ALU multiplier, as well as shifter.

(Refer Slide Time: 49:27)



SHARC Features

- Load-store architecture
  - Operands in register before operation
- Two data address generators
  - Program memory, data memory
- Special loop instruction : zero overhead
  - Loop counter stack for nested loops
  - SHARC has a PC stack – 30 deep for storing subroutine return as well as loop addresses

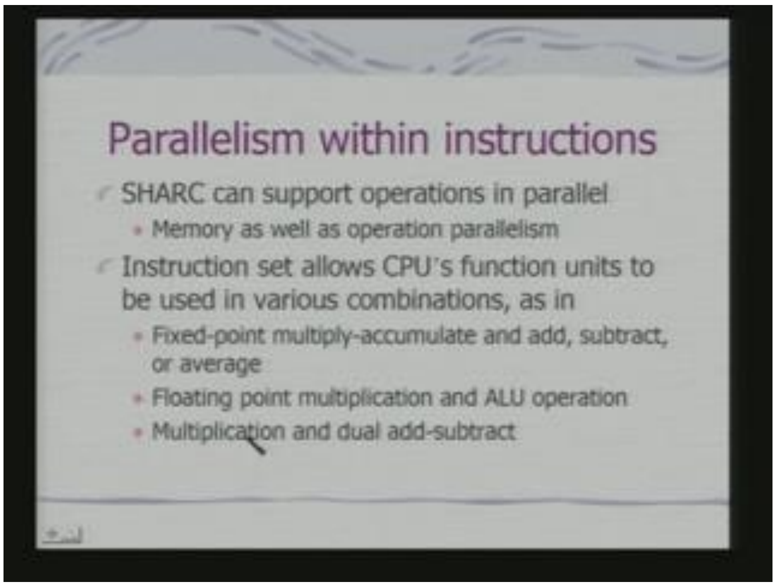
And it is a more RISC like processor. So, it implements a load stored architecture. It has got two addresses, data address generates for the program memory and for the data memory. Just like any other processor, it also implements special loop instructions and zero overhead. That let us look at SHARC for implementation of this special loop instruction. Because, this gives a particular case study, how it can be implement. So, you have got a loop counter stack for nested loop.

So, if I have a loop counter. And there are nested loops, what is that mean? The inside loop has to be first finished, before I going to the outside loop. So, the loop counts have to be put into stack. So that in hardware, I can keep track of the nested loop. Because, it is not happening in software, in the hardware itself by an single instruction you are telling that, following sequence of instructions have to be executed of fix number of times. And that set of instructions can have loops in itself.

So, I need to have a stack of loop counters. So, this is a hardware stack, which is made available for implementation of nested loops. Also it has got a PC stack. In fact, we talked about the TI 55, where we say it is a software stack implemented in the memory. But, SHARC has got a PC stack, it is 30 deep for storing subroutine return, as well as loop addresses. I hope you understand, what loop address is. The beginning of the loop, where we come back to the loop, that has to be stored in the form of a stack in the PC.

So, this gives a particular implementation scheme, in a DSP for loop zero overhead loop, as well as the subroutine call. And this is what I want to discuss distinct for SHARC. SHARC is a VLIW processor.

(Refer Slide Time: 51:44)

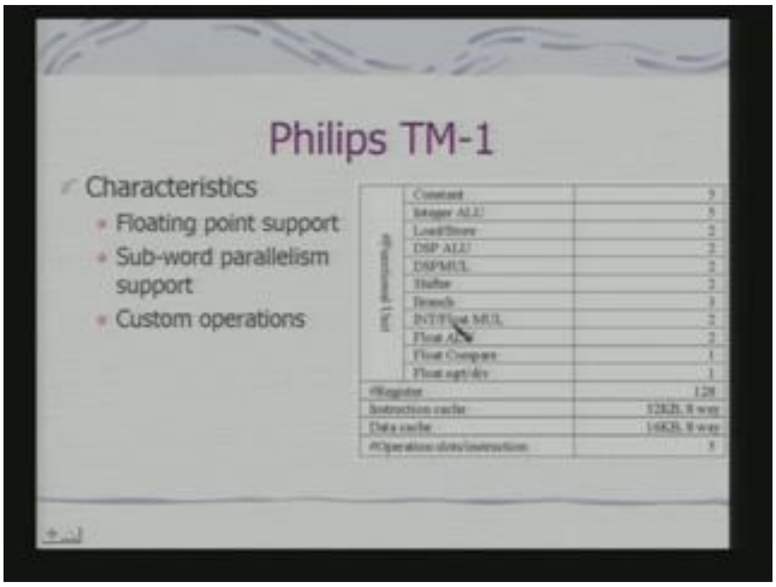


Parallelism within instructions

- SHARC can support operations in parallel
  - Memory as well as operation parallelism
- Instruction set allows CPU's function units to be used in various combinations, as in
  - Fixed-point multiply-accumulate and add, subtract, or average
  - Floating point multiplication and ALU operation
  - Multiplication and dual add-subtract

And it implements a number of instructions in parallel and here you can have various combinations. You can have fixed point, as well as floating point combinations. Obviously, you can realize the internal hardware for SHARC could be more complex, because it is supporting floating point operations. And as I have already discussed earlier, that floating point operation implies what, additional power consumption as well.

(Refer Slide Time: 52:11)



Philips TM-1

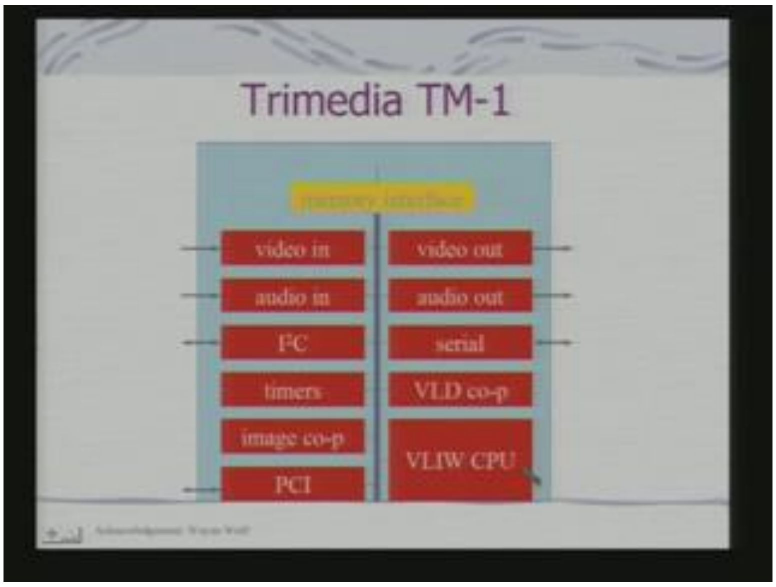
- Characteristics
  - Floating point support
  - Sub-word parallelism support
  - Custom operations

#Functional Unit	Constant	5
	Integer ALU	5
	Load/Store	2
	DSP ALU	2
	DSPMUL	2
	Shifter	2
	Branch	3
	INT/Float MUL	2
	Float ALU	2
	Float Compare	1
	Float sqrt/div	1
#Register		128
Instruction cache		32KB, 8 way
Data cache		16KB, 8 way
#Operation slots/instruction		5

There is another processor, which will like to look at is Philips TM because, this is a processor which has been used wildly for multimedia processing. And this is again a VLIW processor, but there are some distinct features for it. It supports floating point operations, just like your SHARC. And it also has some more customized operations, for this media processing tasks. In fact, it is a much more elaborate, in terms of hardware architectures. And it is particularly intended for media processing.

(Refer Slide Time: 52:54)

Trimedia TM-1



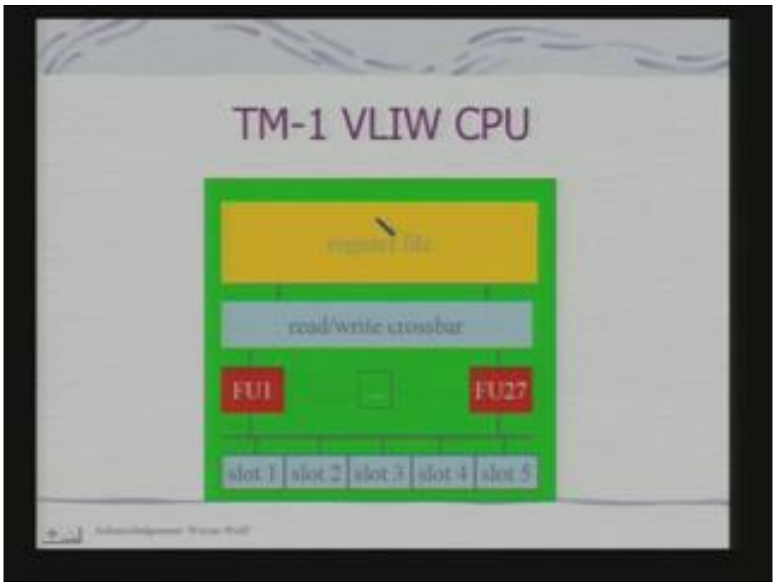
Acknowledgement: Wayne Wolf

So, if you look at the architecture, you will find this is I have got a VLIW CPU. But, along with VLIW CPU, there are specific coprocessors. In fact, this is also a trend for various dedicated DSP’s, which are getting designed. So, these coprocessor is particularly for image processing. And this is also for various VLD coprocessor, this is also variable length coding application, likes a Huffman coding application.

This can be used for implementation of Huffman coding. And it has got specific peripheral blocks, for enabling video in, video out, audio in and audio out. And apart from the expected memory interface, which is common with other DSP’s as well. So, these are example of a crime media, this is the Philips media processor. That when we are deciding a DSP for a particular task, your choice of interfacing blocks can become different.

So, in this case you have got the interfacing blocks, which are targeted for video and audio and you may also have coprocessor, we have discussed the concept of coprocessors with ARM. And we looked at there may be special instruction, in the ARM family itself, which has the instruction targeted for coprocessor. Here, we have seen that in the same silicon area itself, we are talking about an image coprocessor, as well as very well. Another coprocessor, which can be used for this coding and variable length compression tasks, and the basic VLIW CPU is this.

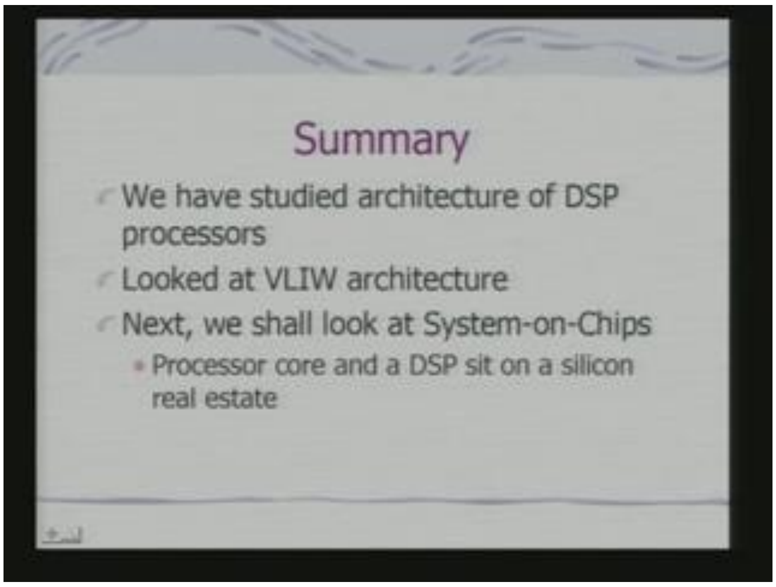
(Refer Slide Time: 54:36)



And this VLIW CPU has got an interesting organization also. So far what we have looked that, we have looked that the same register file or partition register file. Now, here there is a single register file, but there is a crossbar switch. Just like any kind of a communication system, we have a crossbar switch; for connecting this functional unit to registers, there is a switch here.

Depending on the instruction, this switch gets program. So, effectively I can establish a part, from a particular register to a particular functional unit. And you will find that the number of functional units, here it is of the order of 27. So, since we are having a large number of functional units here. So, I really cannot have register file separated and provided separately because, there may be different demands. So, there is a crossbar switch, which is providing these kind of a communication.

(Refer Slide Time: 55:46)



Summary

- We have studied architecture of DSP processors
- Looked at VLIW architecture
- Next, we shall look at System-on-Chips
  - Processor core and a DSP sit on a silicon real estate

So, this finishes our discussions on DSP’s. We have studied primarily in the architectures, some features of their instruction set. We have looked at VLIW. And we move that why VLIW is a kind of a preferred architecture, because there as to be a number of instructions. The number of instructions can be executed at different, on different data items. And VLIW architecture is being adopted in a variety of processor, which are getting implemented.

But, one issue that I have not discussed is that, how to generate the combination of instructions, which are to be executed in parallel. If I take a simple example of C55. If you are doing an assembly language programming, you may put instructions in parallel explicitly. And then, check whether there is any kind of resource conflict, of a assembling and running on it on a simulator. But, that becomes a complex manual process so, today's compliers, which are targeted for these processors.

They are provided with some of the special logic to detect this parallel possibilities. That is from your code itself, whether this parallel possibilities can be extracted. And accordingly, whether the low level machine instructions can be assemble together, to exploit the other level hardware, to the maximum possible extend.

In fact, there are also responsibilities, which come to the program or as well. So that, compilers can easily detect this features and generate efficient code. In the next class, we shall look at what we call system on chips; where a processor core and the DSP, may sit on the same silicon area itself, for a dedicated task. There may be other combinations as well because, the system on chips becoming very important for embedded systems. Because, if you get once such system on chip, all the functions can be mapped onto it and the complete system maybe built around it. Any question?

Student: ((Refer Time: 58:05))

The question is how is it that, we are having a zero overhead loop interrupt architecture. See the basic idea for zero overhead is, that when you are executing a loop in the software, you actually check that whether you are the number of times. Let us consider that I want to executes, this a following three instructions five times. So, I shall have a counter stored in some register, I shall decrement the counter using a specific instruction. And check whether it has becomes 0 or not to terminate the loop.

So, there is an instruction overhead, these are overhead instruction for implementation of the loop. But, if that entire thing can be built into the hardware that means, I say that following set of instructions to be executed five times. So, every time instructions gets executed, this decrement take place on the check takes place, in the same instruction cycle itself. So, effectively you are not having any instruction cycle overhead, for implementation of the loop.