Signal Processing Algorithms & Architecture

Dr. Anirban Dasgupta
Department of Electronics & Electrical Engineering
Indian Institute of Technology Guwahati

Lec 19: Introduction to Signal Processing Architectures

Hello everyone, welcome to a fresh new lecture on the topic of introduction to signal processing architectures. This is Dr. Anirban Dasgupta and let us get started. So, what do you mean by a signal processing architecture? So, in general, what do you mean by architecture? Suppose you are designing a building; what do you mean by the architecture of the building? It means how many bedrooms there will be, how many kitchens there will be, not only how many bedrooms and kitchens or how many bathrooms, but exactly where you would like to have the bedrooms, bathrooms, or kitchen, and what the size of each bedroom, kitchen, and hall room should be, etc. Similarly, signal processing architecture typically refers to the structural organization of both hardware and software systems to process signals. And why do you need architecture? You can just plug in anything, and will it not work? That might work.

But like in a room, if you keep the bedroom of any size and the kitchen anywhere, that will or might work. But you should have a better design for optimal use. Similarly, in signal processing architecture, you should have these elements organized such that they are optimized for tasks such as filtering, compression, enhancement, and any analysis of signals, which are basically the different signal processing tasks. And this is important in building efficiency.

So, what do you mean by efficient? Maybe efficient in terms of cost or space and scalability, which means that once you create this architecture, you can repeat this process and make multiple copies of the signal processor. One example is the video that is recorded using some architecture that is efficient for compressing the video so that you get the best quality video with optimal use of resources, or say, minimum cost. So typically, these are the components in a signal processing system, and it starts with sensors, which convert the physical quantity to the signal or electrical quantity. That electrical quantity is conditional. Conditioned means that this sensor can produce a signal that is either a current or a voltage.

Now you cannot directly send this to an ADC. ADC takes voltage readings. So, if it is a current signal, then you have to convert this current to voltage using an I2V converter like an OP-AMP. Then this value of the voltage may be in microvolts or in millivolts. but this ADC may accept something like a 3.3-volt or 5-volt range. So, if you send this level,

then everything will be logic 0. So, you have to amplify the signal. Also, there can be some criteria for Nyquist sampling. So, you need to have an anti-aliasing filter.

So, all these are performed using signal conditioning, which typically uses some analog circuits. Now, once we get the signal ready, or the conditioned signal that is sent to my ADC. Now, this ADC can be outside the signal processor, or it can be part of the signal processor, and this converts the analog signal to my digital signal stream. Now, this is a process; process means it can be anything like filtering, compression, enhancement, or whatever you feel like, and then the data can be stored in the memory. You can directly send the data from the ADC to memory using something called direct memory access, bypassing the processor, or you can use the stored result in the memory; for example, when my video is being recorded, it is processed and stored in a compressed format in the memory.

Or you can transmit to a digital-to-analog converter, which will convert it to a real-world analog signal. And then this actuator will convert this analog signal, which is an electrical quantity, to any physical quantity or may take some physical action. One example is the speaker, or it can be a robotic arm. So, there are several types of signal-processing architectures. The first is a centralized architecture, which means there is only one system that is performing the signal processing task.

Of course, this is burdened with everything because it is only performing the task. So, this is typically used when the complexity of the algorithm is low, for example, basic filtering or convolution. When the complexity increases or the signal length is very large, you can opt for a distributed architecture where there are several processing units that work either in parallel or in series, but they divide the work like a divide-and-conquer approach. Then parallel architecture, now parallel architecture, can be centralized or distributed, but the main aim is that it can use multiple processing units that operate simultaneously, and these are built using multi-core processors and graphics processing units. Now, GPUs are becoming very popular with the increase in deep learning algorithms.

Then FPGA, which are field programmable gate arrays. I will explain what this is. So, you can divide a big task into smaller tasks which can be executed in parallel. And some examples are single instruction multiple data, where, say, one instruction like convolution can be applied to multiple data in parallel. or multiple instruction multiple data MIMD.

Another example is pipelined architecture, where your processing will be serial, but your task will be divided into several stages, and it will occur in an overlapped fashion. It means that when one instruction is being executed by stage one, the other instruction can use stage two and finally, the hybrid architecture, which combines everything. It can have

centralized ability, distributed ability, parallel ability, and pipelined ability, among others and it can be used in extremely complex tasks that can utilize cloud systems, cloud computing-based signal processing, or even advanced communication systems.

So, these are the typical types of signal-processing architectures. Now, what are the signal-processing components? This diagram I showed you at the beginning defines what sensors are that will capture the real-world signal, which is any signal that appears in the form of a physical quantity and is converted into an electrical quantity at the output of the sensor, typically a current or a voltage. Then signal conditioning modifies the analog signal to have a voltage and frequency range that is suitable for digital conversion, and it is sent to my ADC, which will generate my digital signal. This digital signal is processed using a digital signal processor, and it need not be a digital signal processor strictly, but here we are focusing on a digital signal processor. But nowadays, people can use anything like microcontrollers, microprocessors, FPGA boards, and memory.

Now, whenever you are doing this processing, like if I ask you to solve a problem such as $19 + 3 \cdot \ 7 \ + \ 17$, it is very difficult to calculate directly by listening to the problem; you might prefer to write it down in some notes. So that is the use of memory for storing the variables I mentioned and for storing intermediate results used for buffering, like when the signal stream is coming; for example, the signal is coming at 100 samples per second, but I am able to process only 80 samples. So, the remaining 20 will be kept in a buffer. Then the digital to analog conversion, which is finally the output of the signal processor, will be a digital signal y[n]. So that needs to be converted to analog for reuse.

But then again, this is an optional part, but many signal processing systems have these digital-to-analog converters and finally, the actuators can perform certain actions based on the output signal and that can be as simple as glowing an LED or sounding something in the speaker, like saying I want to have an alarm that, if there is an anomaly detected, then it should sound an alarm that yes, there is some anomaly. So that is an actuator. So we will start with sensors, and I will not go into detail about sensors because this is not a sensors topic, but in general, today all the sensors that are available, or what you can see on the screen, are compatible with many microcontroller boards like Arduino or ESP.

As well as in signal processing boards, most of them are digital sensors. So they have the signal conditioning and the ADC built into the sensor itself. In some cases where this ADC can be separate, you have to either implement a separate ADC for interfacing or directly integrate it with a DSP or a processing unit that has an ADC. And that is the next topic, which is analog-to-digital converters. So, ADC typically has three stages: sampling, quantization, and encoding, and that is very easy to remember.

So, what is sampling? Sampling is when we have the signal, and we have explained it. So, this is the analog signal in time, and this is a voltage varying with time. So sampling is discretizing the time axis, and this is typically done at a fixed rate. There are some samplers, or I would say ADCs, that can do non-uniform sampling, but for simplicity, let us assume that it is a uniform sampling system, and this is specified by the sampling rate or sampling frequency given in Hertz. The next step is quantization, which is discretizing the y-axis or the amplitude axis.

So, quantization is specified first by bit size, which indicates how many levels I want, and the range of the input voltage level. For example, if my quantizer is 0 to 3.3 volts and it is a 2-bit quantizer or 2-bit ADC, that means it can have four values. 0, 1, 2, and 3 or 0, 0, 0, 1, 1, 0, and 1, 1. So, it will divide this range from 0 to 3.3 into 4 levels, and each will be assigned one value, like from 0 to this value; this amplitude will be assigned, say, a value of this, and then this is assigned this. Something like this. Now this is one way of mapping; it is not necessary that it has to be mapped always in this manner. The final step is encoding because you get a quantized value, which is one of these levels that has to be finally sent as a sequence of bits that is this n-bit. So, when you want to buy an ADC, you have to provide these three specifications.

One is the sampling rate, the second is the bit size, and the third is the input voltage level. So, here we can see that we have four bits. So, it can represent anything from 0 to 15, 16 possible levels like 0000 to 1111. Let us take an example to make this clear. So, say I have a signal, which after sampling, gives me these values: 1, 2, 3, 4, 5, and 6.

Very well. So let us say this ADC is a 3-bit ADC, which means that we have 8 levels, 2 to the power of 3 and if the voltage range is, say, 0 to 5 volts, what will be done? So, in the quantization, we have to divide the range. So, what should the range be? The range is

$$\Delta = \frac{V_{max} - V_{min}}{2^n}$$

which means this is 0 to 5 and 2^n . n here is the number of bits, which is 3.

So 2^3 is 8 and this is the value. This can also be called the resolution of quantization or the step size. Now these are the signal values that I mentioned, these values. So, if you see, now if I say this is my range from 0 to 5 and I have divided it into how many? So anything between this.

So, 0.625 was the resolution. So, anything within this may be mapped to a value of 0 like this. This is mapped to zero. So, this will be mapped to 5. This will be mapped to that.

This will be mapped to that. So, in total, I can have 8 values. If you use a 4-bit quantizer, then there will be 16 possible values. So, the more bits you are using, the finer approximation you are making to the analog signal and finally, you know that these values, even if they are quantized, cannot really be put in the hardware because they are stored in registers, which are basically made up of flip-flops and can store either a high or a low, on or off. So, here there are 3 bits: say this is my bit 0, bit 1, and bit 2.

So, I have to use either of these combinations. So, there are 8 possible combinations. So, 5 will be mapped to 111. Similarly, this will be mapped to 110, and so on. Now next coming to memory. So, as I said, when I am discussing anything with you or when you achieve any new information, it is very difficult to keep in mind or memorize things. Of course, the mind is also a memory, but often we tend to write it down. So, whatever you are keeping in mind, this is a kind of working memory or a kind of RAM. And something that you note down and want to keep for a very long time is kind of a permanent memory or ROM, a non-volatile thing. So, I am not saying that the temporary memory you are recollecting cannot be a long-term memory, but often we keep it in a working memory phenomenon or a short-term memory.

Like, if I go to a movie and I need my seat number, I will remember it just until I finish watching the movie, and after that, I can forget it. So, similarly, memory can be used for program instruction, where the program code, which is the set of instructions, will be saved, and this is typically in the ROM or flash memory, which means permanent. So, I mean the signal processing system will perform convolution on whatever data is coming. Then input output data, and these are typically variable data; like now, the input is different, and after 1 hour, the input will be different. So, this memory should typically be temporary and stored in memory buffers and queues, and similarly, the output of the result.

So, these are saved in temporary memory. And this temporary memory is often in the form of RAM. You can also store them in some other kind of memory, which is not totally non-volatile, but that is entirely optional. And then immediate results to store processing of immediate results, like I said, the 2 + 3 + 7 + 17. So together you may not be able to add all four operations. So, maybe you will do 2+3 first, then you will add the result to 17, something like that.

So, that is what the primary and secondary are. So, secondary can be both external and internal. Primary is typically internal, but primary can be external as well. External and internal here mean whether this memory is within the microcontroller or DSP, which is

register memory, or whether external means I have an external RAM or ROM that may again be interfaced with the processor or can be inbuilt. And that brings the concept of processing units where we want to do the operations. And these operations are typically a set of programs.

As I said, this is an advantage of DSP over ASP: I can just change the program, and that changes my signal processing operation. And where do I store the program? So typically, I write the program on a computer and transfer it through some cable or other media to the microcontroller or microprocessor memory. Program memory, you would say, makes things like microprocessors, microcontrollers, and DSPs, where the main difference is that a microprocessor is a chip that contains your major operational units, such as your arithmetic logic unit (ALU), control unit (CU), some registers, and some internal memory, etc. But you need external things like RAM and ROM to work with. And this is typically found in our computers, and you have to interface with this; otherwise, it will not work.

So, the good thing is that you can easily change your RAM size depending on the application. But the bad part is that your physical size will be larger. But in microcontrollers, everything is in the same IC. So, this is a microprocessor, and in DSPs, there are some specialized hardware like barrel shifters and MAC hardware that are very dedicated to signal processing tasks. And finally, we have the FPGA, or field-programmable gate array.

Now this is a different concept altogether because in this by changing the filter you are just changing the code. That means the content of the registers of your program memory are just changing. So, if you write a separate code, it will work as a separate filter. But in field-programmable gate arrays, or FPGAs, all the hardware connections are changed when you reprogram them. So, you have to use some HDL, or hardware description language, to code it into the FPGA.

Altogether, the hardware connections are changed for this FPGA, and that is why it is called field programmable. But the architecture of this thing will be the same; only the content of the program memory will change. So here we are more interested in learning about the signal processors, DSPs. So, as the name suggests, these are specialized and customized for signal processing applications. And there are three things that are taken into account for designing DSPs.

First is real-time processing, which means that it should work within real-time constraints. For example, if you are shooting a video and your audio and video are not synced, that will destroy your purpose. So that is real time processing constraint that should be there. It should work within the time or say you are clicking a photo of a bird that is sitting, and the moment the capture is done, the bird has flown away.

So your purpose is gone. then high throughput that means it should execute maximum number of instructions maybe in parallel if possible wherever possible do it in parallel and low latency. Latency is basically the difference in time between when you give the command and when you actually start doing it. So, that is the main difference: throughput means the rate at which data is processed. For example, if a processor can execute 1 million instructions per second, its throughput is 1 MIPS. Whereas latency is the total delay from when the command is sent to when an operation starts.

For example, if I want to load data, I send the command load. So, this command is issued, say, at this interval at $t = t_0$; this command is sent to load the data, and it starts loading at t1. So, this difference in time $t_1 - t_2$ is my latency. It is as if somebody is commanding me to do the job, and the moment I start doing it, that gap—understanding that thing—is latency and that involves a lot of things like traveling paths and delays in decoding, etc. So, what are the techniques to enhance computational throughput? So, throughput should be high, and latency should be low.

So, there are some specialized modifications in digital signal processors. Now these often come to microcontrollers as well, and the first is a dedicated multiplier, which is specialized hardware for fast multiplications. Then split ALUs so that two or more ALUs work in a simultaneous fashion. Another thing is Harvard architecture. Now this is an architecture where we have the data memory and the program memory separated.

The separate memory for instructions and data. And then there is parallelism. Parallelism means executing multiple operations simultaneously. Another thing is pipelining, which I discussed as overlapping instruction execution with stages. And then there is also something called a barrel shifter. A barrel shifter is like a circular shifter that can shift not only a single bit but multiple bits in one clock cycle.

So let us understand what these are. But before these, here are some examples of real-world DSPs. So, this is the Texas Instruments TMS320 series. Then the Analog Devices ADSP series, then NXP Freescale DSPs, and Qualcomm Hexagon DSPs. So, what is Harvard architecture? So, in Harvard architecture, as I said, the program memory and the data memory are separate. What is the program memory? Program memory means the set of instructions, such as move this, then call this, load this, add this.

So, these instructions are kept in registers, and that is in the program memory where you dump the code. Data is where you get information, such as from sensors, or data that can be generated, like random numbers. So, this is data memory. So, Harvard architecture is like having separate buses. So, the data bus and address bus are separate, and even the control buses are also separate, but I have mostly seen them referred to as the data and address bus, and this is the main processing unit: the ALU and the control unit.

Whereas in Von Neumann architecture, which is used in our PCs, there is a single data bus and address bus. Also, you see that this data bus is bidirectional. That means data can come and go from the processor to the peripheral, whereas the address bus is just from the processor. It is not to and fro. It is like saying you have a big cupboard and only one shelf, and you keep your jeans and shirts on the same rack.

So, that is your Von Neumann and it may be difficult or slower to access. But at Harvard, since they are separate shelves, you can easily access them, and you can access them in parallel, like simultaneously. So, these are the summaries of the differences; in Von Neumann architecture, you have a single memory that is shared, and you have separate memories in Harvard architecture. So, the data and instruction path in Von Neumann is the same bus, and in Harvard, there are separate buses for parallel access. So, Von Neumann can be slower because of the shared bus, whereas Harvard is faster. And Von Neumann is naturally simpler and less complex, but Harvard is more complex in design. So, thank you so much. We will meet again.