IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

hAirLenGth Chainage Storage Wage (GLH)

[1]Mr. Buddh Bhagwan Sahu, [2]Assistant Professor

Dept. of Computer Science & Engineering, Columbia Institute of Engineering and Technology,
Near Vidhan Sabha, Tekari-493111, Raipur (C.G.), India[3]

CHAPTER-01

Abstract: Hair Length Technology is one of the techniques of storage of data in the form of computer language. It will store the data as like a hair length. Dot bounded or chainage methods it’s stabled in the one small point and can be able to store more data from the other ROM or Storage devices. The mapping and storage is a major different from the other methodology that technique is more challenging and cannot be defeat by any other formula at that time. The digital data will be stored in the form of hair grow. Hair grow on the route like data will be stored on the route occupied less point spaces & can be stored big data. The basic understanding of how software functions is helpful for anyone who interacts with technology. With a background in programming, you can get an implantation by coding, designing software, data architecture, or creating intuitive user interfaces.

Keywords: Procedural programming languages, Functional programming languages, Object-oriented programming languages, Scripting languages, Logic programming languages, Front-end language, Back-end languages.

INTRODUCTION

While you'll find dozens of ways to classify various programming languages, they generally we are describing and implementing using following key and programming languages that will be PPl, FPL, OOPL, SL, LPL, FeL, BeL etc.. A procedural language follows a sequence of statements or commands in order to achieve a desired output functional languages focus on the output of mathematical functions and evaluations [5].
This type of language treats a program as a group of objects composed of data and program elements, known as attributes and methods. Objects can be reused within a program or in other programs. Scripting languages to automate repetitive tasks, manage dynamic web content, or support processes in larger applications. Some common scripting languages include [7].
A logic programming language expresses a series of facts and rules to instruct the computer on how to make decisions. The front end deals with all of the text, colours, buttons, images, and navigation that the user will face when navigating your website or application. Back-end languages deal with storage and manipulation of the server side of software. By combining all of the above methods we can implement Hair Length Chainage Storage wage.
The optimization of distributed-ROM-based finite state machine (FSM) implementations as an alternative to conventional implementations based on look-up tables (LuT). In distributed-ROM implementations with constant output value is called constant look-up tables and LuT’s with the same content is called equivalent look-up tables can be saved [6]. A popular test of working memory is the complex span task, in which encoding of memoranda alternates with processing of distracters. A recent model of complex span performance, the Time-Based-Resource-Sharing (TBRS) model has seemingly accounted for several crucial findings, in particular the intricate trade-off between deterioration and restoration of memory in the complex span task working memory has often been characterized as a system for the simultaneous maintenance and processing of information [4].
This working definition is reflected in the complex span paradigm, which has become the most popular method for psychometric measurement of working memory capacity

And which also serves in many experimental investigations of working memory processed in reading span, participants read a series of sentences and try to remember the last word of each sentence. The number of sentences in each series is gradually increased, and a participant's span is determined as the maximum number of sentence-final words they can recall correctly in the order of presentation and capping more size of stage space.

It is not a solution to increase its storage capacities why we should not try to increase its functional medium. In this research we are giving theory how to increase storage in the different medium i.e. hAirLenGth Chainage Storage Wage [9].

DISTRIBUTED-ROM-BASED IMPLEMENTATION

The (The time-based resource-sharing theory) theory makes the following basic assumptions: Representations in working memory decay over time, but they can be refreshed by directing attention to them. Attention is conceptualized as a domain-general mechanism that can be devoted to only one process at a time, and hence creates a bottleneck. In tasks like the complex span task, the cognitive system must devote attention to carrying out each step of the processing task interleaved with encoding of the memoranda [3]. In between processing steps, however, the attention mechanism can be used to refresh memory items. Thus, during each processing period the attention bottleneck is assumed to rapidly switch between carrying out a processing step and refreshing one or more memory items. Each series of steps is called a procedure, and a program written in one of these languages will have one or more procedures within it. The architecture of the time-based resource-sharing theory is a two-layer connectionist network with one layer dedicated to the representations of positions and the other to the representation of the items. The two layers are fully interconnected, and their weights initialized at zero and commonly use of procedural languages includes [9]:

1. C/C++
2. Java
3. Pascal
4. BASIC

Each function–a reusable module of code–performs a specific task and returns a result. The result will vary depending on what data you input into the function. For calculating the weight changes for response suppression, only the activation of the item selected for output is maintained in the item layer on popular functional programming languages include [2]:

1. Scala
2. Erlang
3. Haskell
4. Elixir
5. F#

This makes it a popular language type for complex programs, as code is easier to reuse and scale. In between encoding of items, a series of processing operations must be carried out like simple arithmetic computations or two-alternative forced-choice tasks common object-oriented languages include [10]:

1. Java
2. Python
3. PHP
4. C++
5. Ruby

Programmers use scripting languages to automate repetitive tasks, manage dynamic web content, or support processes in larger applications. To calculate the necessary variables for the simulations, we start from an estimate of mean processing duration for a given experimental condition scripting languages include:

1. PHP
2. Ruby
3. Python
4. Perl
5. Node.js

We do not use the mean duration as the duration of each individual processing step, because we want to simulate a distribution of durations of attention capture, analogous to the distribution of durations of other processes A logic programming language expresses a series of facts and rules to instruct the computer on how to make decisions like examples of logic languages include [11]:

1. Prolog
2. Absys
3. Datalog
4. Alma-0

TECHNICAL IMPLEMENTATIONS

The complex span paradigm requires many modelling decisions about which processes occur when. Among the most important initial decisions is a consideration of the nature of task switching in a complex span task. Any complex span task involves at least two tasks, namely encoding of the memoranda and processing of the distracters. Each item peaks when it is refreshed and drops again while other items are refreshed [12].

Occasionally, an item fails to be retrieved for refreshing and drops out; the gradual decelerated decline illustrates the effect of decay when unbridled by refreshing.

After the final burst of processing operations, items are recalled and then suppressed (which can reduce their strength to be below baseline; thus the beginning of recall can be identified by in forward order, resetting to the first item whenever refreshing was interrupted.

Schedules selective-refreshing scheme would still require a stepping through the list to retrieve the items in order before their strength can be ascertained [15]. Thus, the processing steps of the selective-refreshing approach are identical to the one we implemented, and its outcome is identical as well, because items whose strength is close to asymptote do not gain strength from the "below-zero" dips of activation). During recall of each item, the remaining items continue to decay. Each trial begins with the encoding of the first memory item by associating it to the first position marker [1].

After encoding of the first item, refreshing can only apply to that item, so refreshing effectively means continued encoding of the first item, which is of little consequence because the item has reached nearly asymptotic memory strength already [14].

During the presentation time of later memory items, refreshing in the remaining presentation time includes earlier items and thereby contributes to counteracting decay of these earlier items. We instantiated a refreshing schedule that started from the first list item and proceeded.

The decision to use this refreshing schedule was not arbitrary but the selection was based on consideration of several alternatives refreshing refreshing whether they are skipped or not. One consequence of cumulative refreshing is that the distribution of refreshing time over memory items across a trial is very uneven. Before the second item is presented, all refreshing is concentrated on the first item. After presentation of the second item, refreshing time is divided roughly equally between the first two items

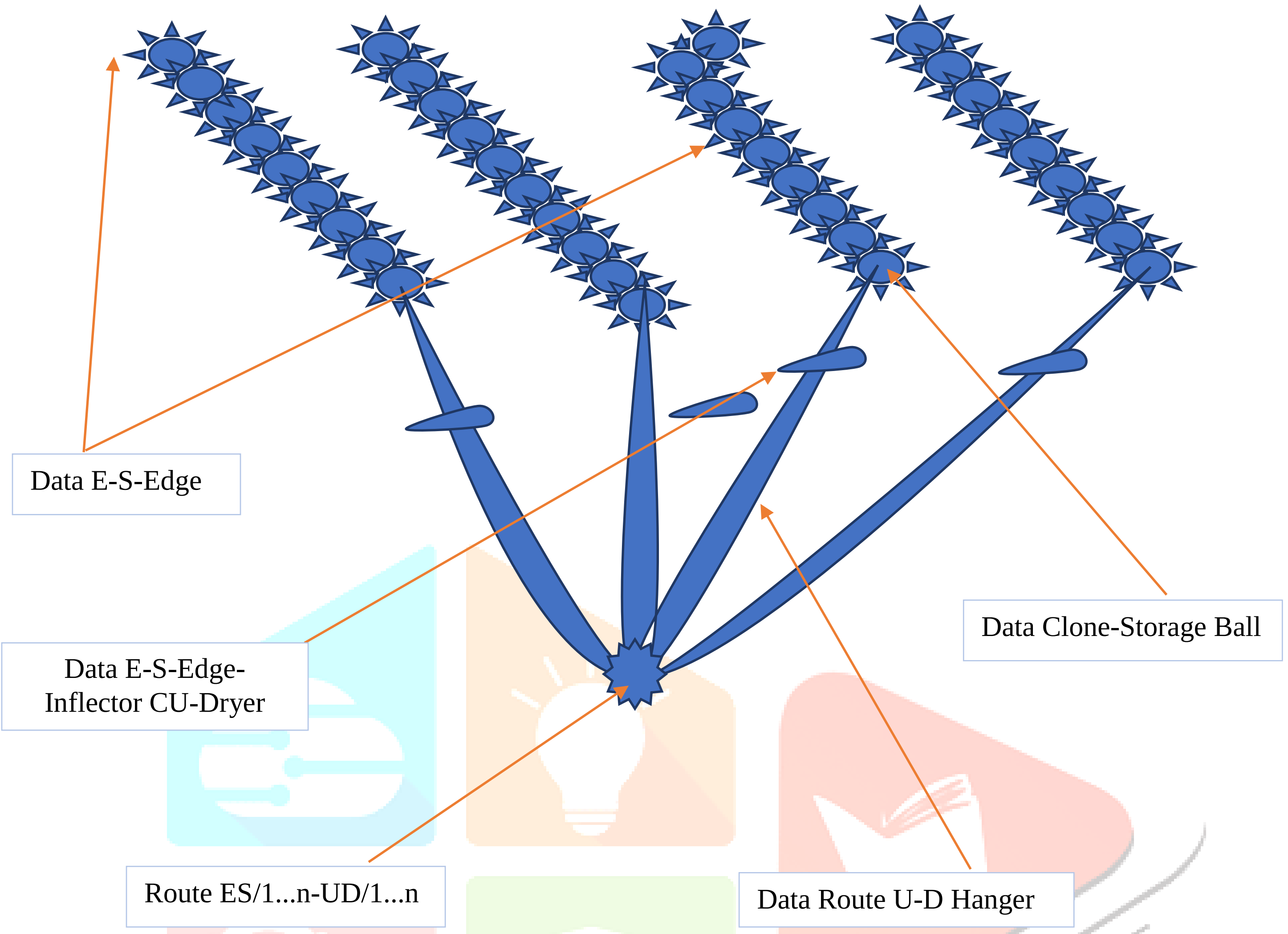

Fig.: HairLenGth Chainage Storage Wage

This memory also displays write-before-read behaviour, common in static memories. What is that? It means that during a write-cycle the data being returned is the same as that being written. Without this, the data returned through the data out port would be the data just being overwritten. Not ideal. This model synthesises into internal block memories in the majority of FPGA architectures. To summarise the key code points from this RAM model, we have data edge to storage edge on end point [16]. That end point will calculate the forwarded data in read mode, read mode will go through the read and write mode to chainage the forwarded data in between route ES/1...n-UD/1----n data edge.
Data edge and storage edge to inflector CU-Dryer (Cluster Up) will host or work as a consortium like sever data. Before releasing the end data it will leave here clone storage ball for the forwarded data point to U-D hanger (Up-Down Data Hanger). For example that VHDL is resolving the many types of data or RAM/ROM/Cache storage technology and its technical implementations using its tools. That is taken for the learning and understanding purposes [13].

```
entity sync_ram is
  port (
    clock   : in  std_logic;
    we      : in  std_logic;
    address : in  std_logic_vector;
    datain  : in  std_logic_vector;
    dataout : out std_logic_vector
  );
```

Unconstrained ports
Std_logic_vector to integer conversion

Write-before-read

End of the classifying the implementation of storage method and clarifying its data signal over data routes. This VHDL post presents a VHDL code for a single-port RAM (Random Access Memory) [18][17]. The VHDL test bench code is also provided to test the single-port RAM in Xilinx ISIM. The RAM's size is 128x8 bit. You also can use many libraries like IEEE, Kaggle, and GitHub etc...

```
end entity sync_ram;
architecture RTL of sync_ram is
   type ram_type is array (0 to (2**address'length)-1) of std_logic_vector(datain'range);
   signal ram : ram_type;
   signal read_address : std_logic_vector(address'range);
begin
```

1. RAM_CLOCK: the clock signal for sequentially writing data to the single-port RAM.
2. RAM_DATA_IN: 8-bit input data to be written to RAM at the provided input address RAM_ADDR when it is enabled.
3. RAM_WR: Write enable signal for writing to RAM, only if RAM_WR = 1, RAM_DATA_IN is written to the RAM at the rising edge of the clock signal.
4. RAM_ADDR: 6-bit Address where 8-bit input data are written to and data are read out.
5. RAM_DATA_OUT: 8-bit output data read out from the provided input address RAM_ADDR.

Logic Design to talk about the implementation of many simply types of RAM in the Hardware Description Language (HDL) that we covered during my series which is VHDL! I promised to do Verilog will come in to understand what I will be talking about I highly suggest you to refresh your knowledge using my series that contains simulating the circuits and even the implementation of more complex units like the ALU that does mathematical operations for CPU's!. RAM is defined as Random-Access Memory and allows us to read and write information to a physical location inside of the RAM which is the so called memory array [20][19].

```
-- A 128x8 single-port RAM in VHDL
entity Single_port_RAM_VHDL is
port(
 RAM_ADDR: in std_logic_vector(6 downto 0); -- Address to write/read RAM
 RAM_DATA_IN: in std_logic_vector(7 downto 0); -- Data to write into RAM
 RAM_WR: in std_logic; -- Write enable
 RAM_CLOCK: in std_logic; -- clock input for RAM
 RAM_DATA_OUT: out std_logic_vector(7 downto 0) -- Data output of RAM
);
end Single_port_RAM_VHDL;
architecture Behavioral of Single_port_RAM_VHDL is
-- define the new type for the 128x8 RAM
type RAM_ARRAY is array (0 to 127 ) of std_logic_vector (7 downto 0);
-- initial values in the RAM
signal RAM: RAM_ARRAY :=(
   x"55",x"66",x"77",x"67",-- 0x00:
   x"99",x"00",x"00",x"11",-- 0x04:
   x"00",x"00",x"00",x"00",-- 0x08:
   x"00",x"00",x"00",x"00",-- 0x0C:
   x"00",x"00",x"00",x"00",-- 0x10:
   x"00",x"00",x"00",x"00",-- 0x14:
   x"00",x"00",x"00",x"00",-- 0x18:
   x"00",x"00",x"00",x"00",-- 0x1C:
   x"00",x"00",x"00",x"00",-- 0x20:
   x"00",x"00",x"00",x"00",-- 0x24:
   x"00",x"00",x"00",x"00",-- 0x28:
```

```
  x"00",x"00",x"00",x"00",-- 0x2C:
  x"00",x"00",x"00",x"00",-- 0x30:
  x"00",x"00",x"00",x"00",-- 0x34:
  x"00",x"00",x"00",x"00",-- 0x38:
  x"00",x"00",x"00",x"00",-- 0x3C:
  x"00",x"00",x"00",x"00",-- 0x40:
  x"00",x"00",x"00",x"00",-- 0x44:
  x"00",x"00",x"00",x"00",-- 0x48:
  x"00",x"00",x"00",x"00",-- 0x4C:
  x"00",x"00",x"00",x"00",-- 0x50:
  x"00",x"00",x"00",x"00",-- 0x54:
  x"00",x"00",x"00",x"00",-- 0x58:
  x"00",x"00",x"00",x"00",-- 0x5C:
  x"00",x"00",x"00",x"00",
  x"00",x"00",x"00",x"00",
  x"00",x"00",x"00",x"00",
  x"00",x"00",x"00",x"00",
  x"00",x"00",x"00",x"00",
  x"00",x"00",x"00",x"00",
  x"00",x"00",x"00",x"00",
  x"00",x"00",x"00",x"00"
  );
begin
process(RAM_CLOCK)
begin
 if(rising_edge(RAM_CLOCK)) then
 if(RAM_WR='1') then -- when write enable = 1,
 -- write input data into RAM at the provided address
 RAM(to_integer(unsigned(RAM_ADDR))) <= RAM_DATA_IN;
 -- The index of the RAM array type needs to be integer so
 -- converts RAM_ADDR from std_logic_vector -> Unsigned -> Interger using numeric_std library
 end if;
 end if;
end process;
 -- Data to be read out
 RAM_DATA_OUT <= RAM(to_integer(unsigned(RAM_ADDR)));
end Behavioral;
```

RAM is a synchronous circuit which means that the information will be stored into it (or the requested send to the output) after a clock event. If the application we have needs to be able to read at the same clock without having to wait for a clock event or even a whole clock period, then we can simply define only a write signal (1 -> write, 0 -> not write) and make the output show us directly what the specified address has, which means that we read all the times, without having to specify that we want to! Keep in mind that this implementation should contain a memory enable signal [21].

1. Data Input -> information to write
2. Data Output -> information to read
3. Address -> we need to know where to read or write
4. Read/Write -> specifying if we want to read or write from an address
5. Enable -> we enable the RAM only when needed to "save power".
6. Clock -> RAM is of course a synchronous circuit and so needs a Clock Input.

Depending on the application we may want to read and write at the same time, and maybe even on different addresses. This can of course be done by having 2 addresses and two separate read and write signals. This means that RAM can also be splitted into these categories:

1. Single-Port RAM with separate read and write signals (no enable needed)
2. Single-Port RAM with a single read/write signal and RAM enable
3. Dual-Port RAM with separate signals (that may be combined R/W signals) for each "line".

We can of course continue on and insert 3, 4 or even more lines as we wish, but I guess that after a point a second or even third memory might be better [22].

```
-- VHDL testbench code for the single-port RAM
ENTITY tb_RAM_VHDL IS
END tb_RAM_VHDL;

ARCHITECTURE behavior OF tb_RAM_VHDL IS

   -- Component Declaration for the single-port RAM in VHDL

   COMPONENT Single_port_RAM_VHDL
   PORT(
        RAM_ADDR : IN  std_logic_vector(6 downto 0);
        RAM_DATA_IN : IN  std_logic_vector(7 downto 0);
        RAM_WR : IN  std_logic;
        RAM_CLOCK : IN  std_logic;
        RAM_DATA_OUT : OUT  std_logic_vector(7 downto 0)
       );
   END COMPONENT;


  --Inputs
  signal RAM_ADDR : std_logic_vector(6 downto 0) := (others => '0');
  signal RAM_DATA_IN : std_logic_vector(7 downto 0) := (others => '0');
  signal RAM_WR : std_logic := '0';
  signal RAM_CLOCK : std_logic := '0';

 --Outputs
  signal RAM_DATA_OUT : std_logic_vector(7 downto 0);

  -- Clock period definitions
  constant RAM_CLOCK_period : time := 10 ns;

BEGIN

 -- Instantiate the single-port RAM in VHDL
  uut: Single_port_RAM_VHDL PORT MAP (
         RAM_ADDR => RAM_ADDR,
         RAM_DATA_IN => RAM_DATA_IN,
         RAM_WR => RAM_WR,
         RAM_CLOCK => RAM_CLOCK,
         RAM_DATA_OUT => RAM_DATA_OUT
       );

  -- Clock process definitions
  RAM_CLOCK_process :process
  begin
 RAM_CLOCK <= '0';
 wait for RAM_CLOCK_period/2;
 RAM_CLOCK <= '1';
 wait for RAM_CLOCK_period/2;
  end process;

  stim_proc: process
  begin
 RAM_WR <= '0';
 RAM_ADDR <= "0000000";
 RAM_DATA_IN <= x"FF";
    wait for 100 ns;
 -- start reading data from RAM
 for i in 0 to 5 loop
 RAM_ADDR <= RAM_ADDR + "0000001";
    wait for RAM_CLOCK_period*5;
 end loop;
 RAM_ADDR <= "0000000";
 RAM_WR <= '1';
 -- start writing to RAM
 wait for 100 ns;
 for i in 0 to 5 loop
 RAM_ADDR <= RAM_ADDR + "0000001";
 RAM_DATA_IN <= RAM_DATA_IN-x"01";
    wait for RAM_CLOCK_period*5;
 end loop;
 RAM_WR <= '0';
    wait;
  end process;

END;
```

In VHDL we will of course only implement static RAM that is build up of memory cells and so creates a memory array.

DISTRIBUTED-ROM ARCHITECTURES

We will use the multiple methods on HairLenGth Chainage Storage Wage (GLH). ROM is Read-Only-Memory which means that predefined data/information (mostly) is written inside of it and we can only read this information and to do so we have to give an address, which corresponds to the point in the storage-array in which the needed data is stored [23]. This makes ROM non-volatile memory because it keeps the information stored "forever", unlike RAM which loses the information on a power loss or reset. So, ROM refers to memory that is hard-wired and cannot be changed after manufacture.

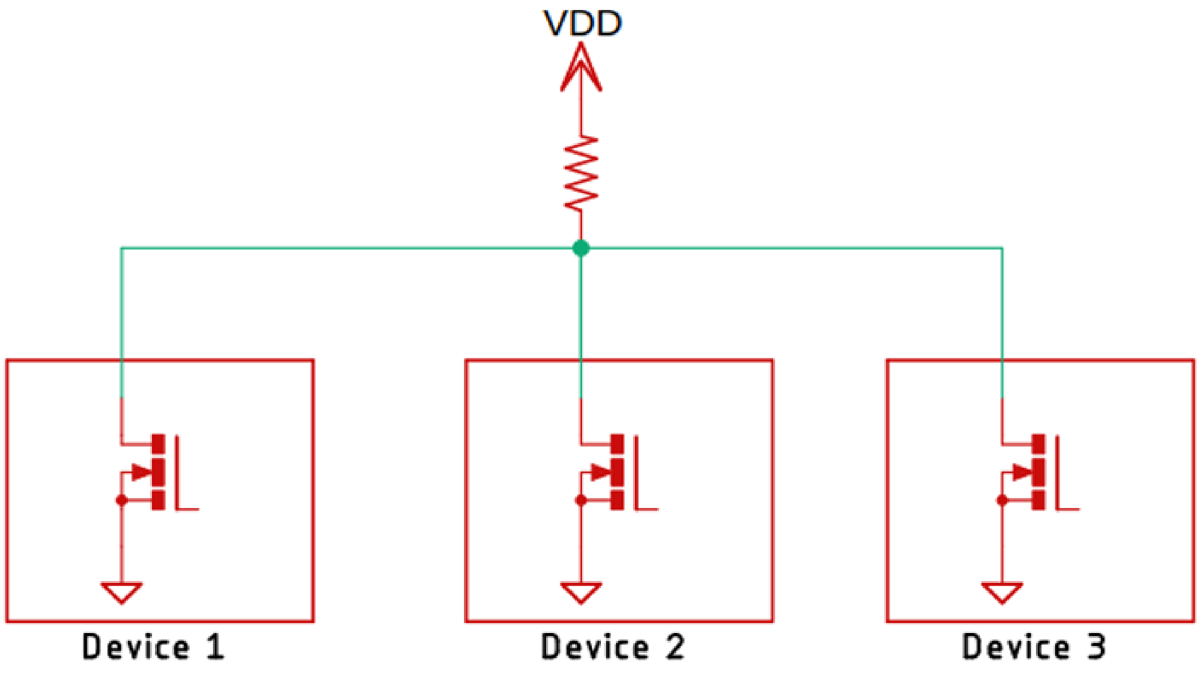

Fig.: Common ROM Architecture.

Modifying the data stored inside of a ROM is very difficult, slow and sometimes not even possible. That's why such memory is mostly used to store firmware which doesn't need frequent updates [24]. Firmware is software directly binded/bounded to the Hardware and mostly contains programs that need to be run at the system start (BIOS). But, restricting ROM from being altered in the manufacturing process makes it useless when an update is required to fix a security issue or bug that's been detected afterwards.
As you can see from the above image, every one-wire device comes in an open drain configuration which means every one-wire device (O-W-D) on the 1-wire bus can pull the voltage to the ground, but no device can drive the bus high and that is why every 1-wire bus includes a pull-up resistor, the resistor value need to be adjusted according to the number of devices that are connected to the bus [25]. A 1-wire device transmits, receives, and provides power to the devices with the same bus so it becomes important to set the resistor value according to the number of devices that are connected to the bus. The resistor value (RV) can vary from a couple of hundred ohms to kilo/h/m/s.

As a 1 wire bus is open-drain there is a MOSFET in the I/O line to pull the bus down when it's time to send data and there is also a buffer B1 to square up the signal in the I/O line. For power, a diode and a capacitor are used. This is the so-called parasite power and with the help of this feature, multiple 1-wire devices can operate simultaneously when the I/O line is low [26].

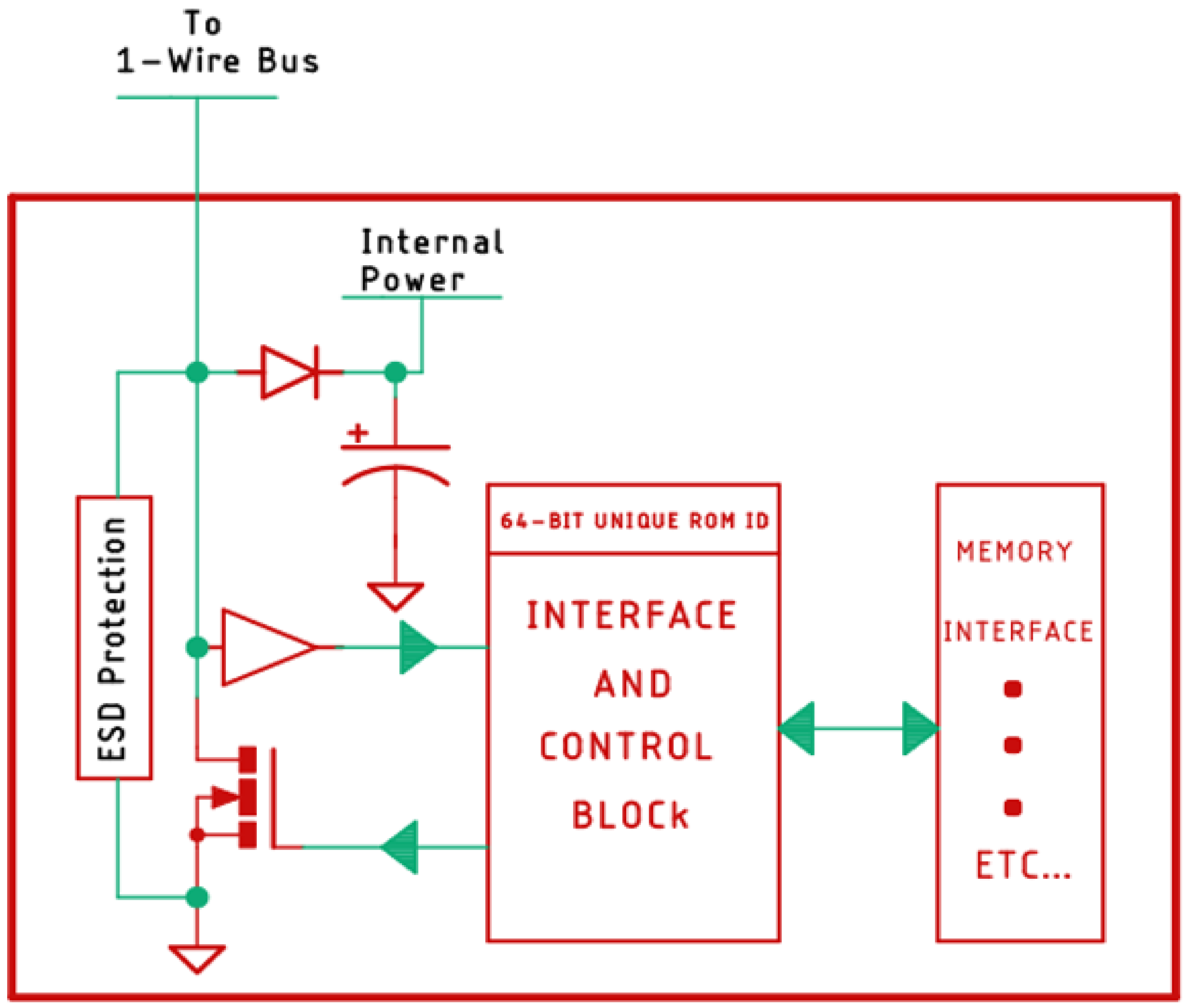

Fig.: Common Internal Power Interface

EXPERIMENTAL RESULTS

We just need to include the required libraries for the Adriano and we can load up the example sketch for that. If everything is connected correctly, we can see the output data on the serial monitor window, but let's just not take the easy way and try to understand how the code works. We initialize our code by including all the required libraries and we define the pin to which the temperature sensor is connected.

```
// Include the libraries we need
#include <OneWire.h>
#include <DallasTemperature.h>
// Data wire is plugged into port 2 on the Arduino
#define ONE_WIRE_BUS 2
```

One handles the one wire protocol and the other one handles the pass the one wire instance through the Dallas Temperature instance [27].

```
// Setup a oneWire instance to communicate with any OneWire devices (not just Maxim/Dallas
temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);
// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);
```

We have our setup () function, in the setup function we enable the serial with the begin method and we print a statement so that we can be sure that the serial monitor is working properly.

```
void setup(void)
{
  // start serial port
  Serial.begin(9600);
  Serial.println("Dallas Temperature IC Control Library Demo");
  // Start up the library
  sensors.begin();
}
```

Next, we have our loop function, in the loop function; we first request data from the sensor with the help of the sensors. Request temperatures () command. We also print statements on the serial monitor window to check if the process was completed successfully or not [28].

```
// call sensors.requestTemperatures() to issue a global temperature
  // request to all devices on the bus
  Serial.print("Requesting temperatures...");
  sensors.requestTemperatures(); // Send the command to get temperatures
  Serial.println("DONE");
```

Next we store the received read data from the local variable named temperature-C and in the next line, we check if it was successful or not. If the data acquisition was successful, we print the temperature on the serial monitor window else we print an error message on the serial monitor window.

```
// We use the function ByIndex, and as an example get the temperature from the first sensor only.
  float tempC = sensors.getTempCByIndex(0);
  // Check if reading was successful
  if(tempC != DEVICE_DISCONNECTED_C)
  {
    Serial.print("Temperature for the device 1 (index 0) is: ");
    Serial.println(tempC);
  }
  else
```

```
    {
      Serial.println("Error: Could not read temperature data");
    }
  }
```

1. Temperature Sensor Reading- Hear is what you should do
2. Check **-ve** and **+ve** leads are connected correctly.
3. Check the Operating voltage of the Device (3.0V to 5V is the normal operating voltage)
4. If you have more Temperature Sensors check the value of the pull-up resistor.
5. If you are interfacing the Temperature Sensor with Adriano check out if you are using the correct library [29].

REFERENCES

[1] Ackerman, P. L., Beier, M. E., & Boyle, M. O. (2005). Working memory and intelligence: The same or different constructs? Psychological Bulletin, 131, 30–60.

[2] Atkinson, R. C., & Shiffrin, R. M. (1968). Human memory: A proposed system and its control processes. In K. W. Spence & J. T. Spence (Eds.), The psychology of learning and motivation: Advances in research and theory (Vol. 2, pp. 90–195). New York: Academic Press.

[3] Baddeley, A. D. (1986). Working memory. Oxford: Clarendon Press.

[4] Barrouillet, P., & Camos, V. (2001). Developmental increase in working memory span: Resource sharing or temporal decay? Journal of Memory and Language, 45, 1–20.

[5] Bhatarah, P., Ward, G., Smith, J., & Hayes, L. (2009). Examining the relationship between free recall and immediate serial recall: Similar patterns of rehearsal and similar effects of word length, presentation rate, and articulatory suppression. Memory & Cognition, 37, 689–713.

[6] Ackerman, P. L., Beier, M. E., & Boyle, M. O. (2005). Working memory and intelligence: The same or different constructs? Psychological Bulletin, 131, 30–60.

[7] Atkinson, R. C., & Shiffrin, R. M. (1968). Human memory: A proposed system and its control processes. In K. W. Spence & J. T. Spence (Eds.), The psychology of learning and motivation: Advances in research and theory (Vol. 2, pp. 90–195). New York: Academic Press.
[8] Baddeley, A. D. (1986). Working memory. Oxford: Clarendon Press.

[9] Barrouillet, P., & Camos, V. (2001). Developmental increase in working memory span: Resource sharing or temporal decay? Journal of Memory and Language, 45, 1–20.

[10] Barrouillet, P., Bernardin, S., & Camos, V. (2004). Time constraints and resource sharing in adults' working memory spans. Journal of Experimental Psychology: General, 133, 83–100.

[11] Barrouillet, P., Bernardin, S., Portrat, S., Vergauwe, E., & Camos, V. (2007). Time and cognitive load in working memory. Journal of Experimental Psychology. Learning, Memory, and Cognition, 33, 570–585.

[12] Barrouillet, P., Gavens, N., Vergauwe, E., Gaillard, V., & Camos, V. (2009). Working memory span development: A time-based resource-sharing model account. Developmental Psychology, 45, 477–490.

[13] Bhatarah, P., Ward, G., Smith, J., & Hayes, L. (2009). Examining the relationship between free recall and immediate serial recall: Similar patterns of rehearsal and similar effects of word length, presentation rate, and articulatory suppression. Memory & Cognition, 37, 689–713.

[14] Bhatarah, P., Ward, G., & Tan, L. (2008). Examining the relationship between free recall and immediate serial recall: The serial nature of recall and the effect of test expectancy. Memory & Cognition, 36, 20–34.

[15] Bjork, R. A., & Whitten, W. B. (1974). Recency-sensitive retrieval processes in long-term free recall. Cognitive Psychology, 6, 173–189.

[16] Brown, S., & Heathcote, A. (2005). A ballistic model of choice response time. Psychological Review, 112, 117–128.

[17] Bunting, M. F., Cowan, N., & Saults, J. S. (2006). How does running memory span work? The Quarterly Journal of Experimental Psychology, 59, 1691–1700.

[18] Camos, V., Lagner, P., & Barrouillet, P. (2009). Two maintenance mechanisms of verbal information in working memory. Journal of Memory and Language, 61, 457–469.

[19] Conway, A. R. A., Kane, M. J., Bunting, M. F., Hambrick, D. Z., Wilhelm, O., & Engle, R. W. (2005). Working memory span tasks: A methodological review and user's guide. Psychonomic Bulletin & Review, 12, 769–786.

[20] Cowan, N. (1995). Attention and memory: An integrated framework. New York: Oxford University Press.

[21] Daneman, M., & Carpenter, P. A. (1980). Individual differences in working memory and reading. Journal of Verbal Learning and Verbal Behavior, 19, 450–466.

[22] Davelaar, E. J., Goshen-Gottstein, Y., Ashkenazi, A., Haarmann, H. J., & Usher, M. (2005). The demise of short-term memory revisited: Empirical and computational investigation of recency effects. Psychological Review, 112, 3–42.

[23] Ecker, U. K. H., Lewandowsky, S., Oberauer, K., & Chee, A. E. H. (2010). The components of working memory updating: An experimental decomposition and individual differences. Journal of Experimental Psychology. Learning, Memory, and Cognition, 36, 170–189.

[24] Zhang, B., Song, S.: Design of CNC oscillator based on field programmable gate array. J. Xi'an Univ. Posts Telecommun. 4, 72–75 (2017)

[25] Kai, Y., Huang, H., Xing, Y.: Design of CNC oscillator circuit in GPS receiver. Inf. Technol. 9, 58–61+66 (2017)

[26] Xue, O., Zou, W.: ASIC design of NC oscillator using look-up table method. Electron. Packag. 8, 13–15 (2012).

[27] Nie, Q.: Design and implementation of direct digital frequency synthesizer based on CORDIC algorithm. Xi'an Univ. Electron. Sci. Technol. (2011).

[28] Li, F.: FPGA implementation of digitally controlled oscillator (NCO). Appl. Electron. Compon. 11, 42–44 (2010).

[29] S.S. Kamate, A. Naikar, S.S. Malaj, Design and implementation of low power flash ADC using cadence tool. J. Adv. Sci. Technol. 12(25), (2016).

BIOGRAPHY

I am **Mr. Buddh Bhagwan Sahu**, B.Tech/M.Tech in **Computer Technology and Application** in Computer Science & Engineering. At present I am working as an **Assistant Professor** at **Columbia Institute of Engineering and Technology** in **Computer Science Department**. I have a more then 8 year teaching excellent experience in Degree cum Technical Academic sector. On behalf of this research paper we are finding the theorem of How to store 1GB data in a 500kb storage capacity without losing its image or graphics. By combining the methodology of Ram and ROM including cache cum cloud tactics it is possible to execute. My basic thought to believe in Innovation, Invention, Research and Relay including Moral, Emotion, Ethics and Rules/Regulation.