PES UNIVERSITY

Karnataka, Bangalore-560080

Course: Quantum Entanglement and Quantum Computing

Course code: UE22EC343AB3

Project on:

“Multi-target Grover-Radhakrishnan-Korepin(GKR) partial search algorithm”

Project by: Purab P Bhat | PES1UG22EC220

Under guidance of :

Dr Kaustav Bhowmick

ECE Dept

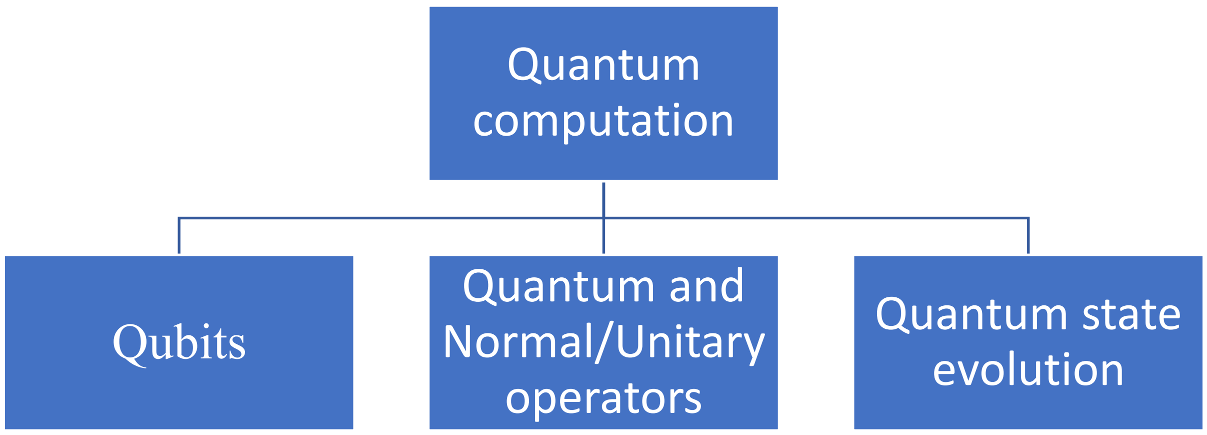
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINNERING

Abstract: In this report we will be looking into oracle based search techniques to find data from a database. The method we will analyse in the report is an optimised version of the Grover- Radhakrishnan based searching model, optimised by V.Korepin. An emphasis on multiple targets is made instead of a single target and can be referred to as multi-target GRK partial search algorithm. The quantum circuit is synthesised using python by importing qiskit and matlab modules and its FPGA implementation is analysed using ZedBoard (Zync 7000).

Introduction:

Quantum computing is a field that leverages quantum-mechanical phenomena to process information. In the early 1980s, Yuri Manin suggested the concept of quantum computations. Around the same time, Paul Benioff established that quantum machines can be used to efficiently simulate classical computers. Richard Feynman further advanced this idea by showing that quantum mechanical systems cannot be simulated efficiently in a classical computer in essence, a quantum mechanical system can only be simulated by another quantum mechanical system. David Deutsch was the first to describe a computational task that can be performed faster using quantum computational resources than by any classical algorithm His quantum algorithm determines whether an unknown function is constant or balanced with just one evaluation using a promise oracle (“black box”). Quantum computing is a field that uses quantum mechanics and algorithms to solve complex problems fast and efficiently. They take advantages of various effects like superposition and entanglement. Oracle quantum techniques also known as “Black Box”. Mathematically, an oracle is a Boolean function that outputs 0 (“yes”) or 1 (“no”). Hence oracle based quantum computing system uses 2 qubits (quantum bits). Deutsch, Shor’s, Bernstein-Vazrani, Simons, Grover’s algorithms are a few to name[1].

In general aspect regarding the mathematics involved in Quantum mechanical modelling, it can be divided into 4 main parts as shown in fig :



a) Qubits- the quantum equivalent of the binary bit used in classical computers. It always exists in a superposition of two states (|0> and |1>). This infers that both, logic 0 and logic 1 can exist simultaineously, which enables parallel computation.

$$0 \to |0\rangle = \binom{1}{0} \text{ and } 1 \to |1\rangle = \binom{0}{1}$$

Dirac notation-

- ket $|\Psi\rangle$ (represents column vector)
- bra $\langle\Psi|$(complext conj of column vector)

b) Quantum and Normal/Unitary operators-The operators are generally represented as matrices.

- $\langle\ |\ \rangle$-Dot product notation
- $|\ \rangle\langle\ |$-Outer product notation

Any operator that satisfies A'A=AA' is a normal operator. Spectral decomposition is one such important consequence, represented as

$$A = \sum i\, \lambda j\, |vj\rangle \langle vj|$$

In addition, if A'A=AA'=I, it is said to be unitary. Quantum computing operators are unitary and hence conserve probabilities.

c) Quantum state evolution- Qubits can be manipulated using gates like Hadamard, C-NOT, Pauli's gate. Gives results indicating if the given function is balanced or unbalanced.

Hadamard gate is defined as

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Controlled-NOT (CNOT) gate is defined as

$$CNOT \equiv CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Grovers Radhakrishnan Korepin (GRK) algorithm-

The GRK search algorithm is an extension of Grover-Radhakrishnan quantum search algorithm, optimized for partial search in databases with multiple blocks or targets. [2] Unlike full Grover search, which identifies a specific item in $O(\sqrt{N})$ queries, GRK focuses on locating a block or subset containing the target item(partial search), enhancing spee. Blocks with target elements are called target blocks, and rest of the blocks without target elements are called non-target blocks. There are B = N/M numbers of elements in each block. Assume that there are K_T target blocks and each target block has B_T number of target elements. So in total there are $M = K_T B_T$ target elements.

Initial state of the vector can be given by:

$$|\tilde{\Theta}\rangle = \sqrt{\frac{M}{N}}|A_T\rangle + \sqrt{\frac{N-M}{N}}|A_{nT}\rangle$$

And the initial probability of obtaining the state is given by:

$$\tilde{\mathcal{P}}_T = |\langle A_T|\tilde{\Theta}\rangle|^2 = \sin^2\tilde{\theta} = \frac{M}{N}$$

angle between the initial state and the normal to the unite target state is given by:

$$\sin^2\tilde{\theta} = \frac{M}{N} = \frac{K_T B_T}{N}$$

The data is divided into blocks and global grover iteration is applied

$$\tilde{\mathcal{G}}^L = \oplus_{\alpha=1}^{K}\tilde{\mathcal{G}}_\alpha .$$

to each block where the target blocks are represented as:

$$|A_{T\alpha}\rangle = \sqrt{\frac{1}{B_T}}\sum_{\alpha\text{block}}^{\text{target elements}}|a_i\rangle, \quad \alpha = 1, 2, \ldots, K$$

And non-target block as

$$|A_{nT\alpha}\rangle = \sqrt{\frac{1}{B-B_T}}\sum_{\alpha\text{block}}^{\text{non-target elements}}|a_i\rangle, \quad \alpha = 1, 2, \ldots, K$$

Only the target block undergoes meaningful transformation. A final global Grover iteration removes amplitudes from non-target blocks, leaving the target block amplified.

Therefore final state alighned with target block is given by

$$\begin{aligned}|\tilde{\mathcal{F}}_T\rangle &= \sin\tilde{\omega}|A_T\rangle + \tilde{\cos}\omega|A_{nTT}\rangle \\ &= \sqrt{M}\left(\mathcal{C}_T - \mathcal{C}_{NTB}\right)|A_T\rangle + \sqrt{K_T(B-B_T)}\left(\mathcal{C}_{NTB} - \mathcal{C}_{TB}\right)|A_{nTT}\rangle\end{aligned}$$

Advantages of multi-target Partial search [3]:

- Faster than Full Search as GRK requires fewer iterations than Grover's full search, especially when the target item can exist in multiple blocks.
- Hierarchical Search of GRK can be applied in multiple levels. After identifying the target block, the block can be partitioned further, and GRK is reapplied to locate sub-blocks.
- If a database of size N=1024 is divided into K=16 blocks, each block contains b=64 items. GRK can identify the target block in O($\sqrt{(\frac{N}{K})}$) queries which is faster than the O($\sqrt{N}$) queries required for full Grover search.

Implementation and Analysis:

A. *Implementation of Multi-target GRK quantum circuit-*

The simulation of the quantum circuit of Multitarget GRK search algorithm can be implemented using python with the qiskit module for quantum bits and gates and computation of quantum data [4] [5]. The circuit can be visualised using plot function of matplotlib library on python. The following results were observed:

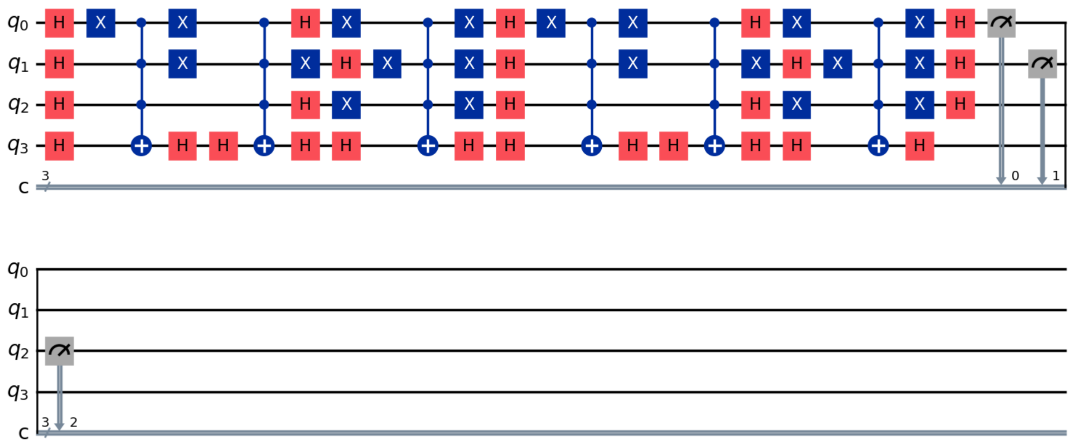


Fig 1a: Quantum Circuit depecting the GRK search algorithm

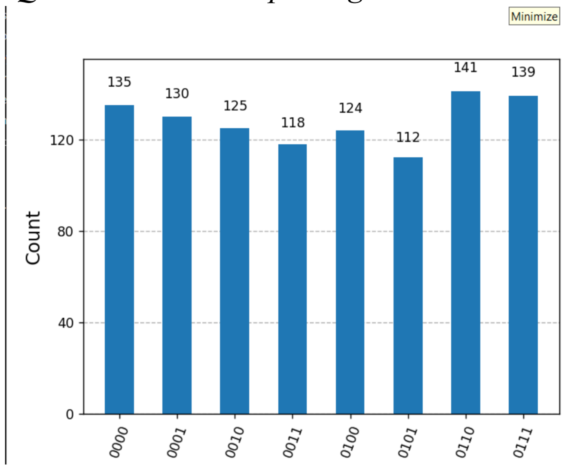


Fig 1b. Histogram to show the count at each iteration(probablity)

Fig1a. shows the circuit which implements the GRK partial search for multi target. This circuit is designed for a 4-qubit system, where the first 3 qubits represent the search space, and the 4th qubit is an ancilla (used for multi-controlled operations).The circuit is structured into three key sections:

- Oracle Implementation: X gates apply conditional flips to represent marked states. Multi-controlled X and Z gates operate on these to encode the oracle logic. A 4-qubit system is initialized with Hadamard gates to create a uniform superposition.
- Diffusion Operator: This is a reflection that enhances the probability of marked states by applying Hadamard and X gates, followed by multi-controlled Z operations. Hadamard, X, and multi-controlled Z gates reflect about the average amplitude.
- Measurement Process: The final measurement collapses the quantum state, and the classical register records the result.

Fig 1b. The histogram shows the results of the quantum circuit after multiple search iterations. The x-axis represents possible output states (0000 to 0111), while the y-axis represents the counts or frequency of each state. Marked states (011 and 101) exhibit higher counts compared to others, indicating the success of the GRK algorithm in amplifying their probabilities. Grover-Radhakrishnan Algorithm (Partial Multi-target Search) enhances Grover's algorithm by focusing on partial marked states. Instead of searching for a single state, it amplifies multiple target states.

B. *FPGA implementation constraints and design*

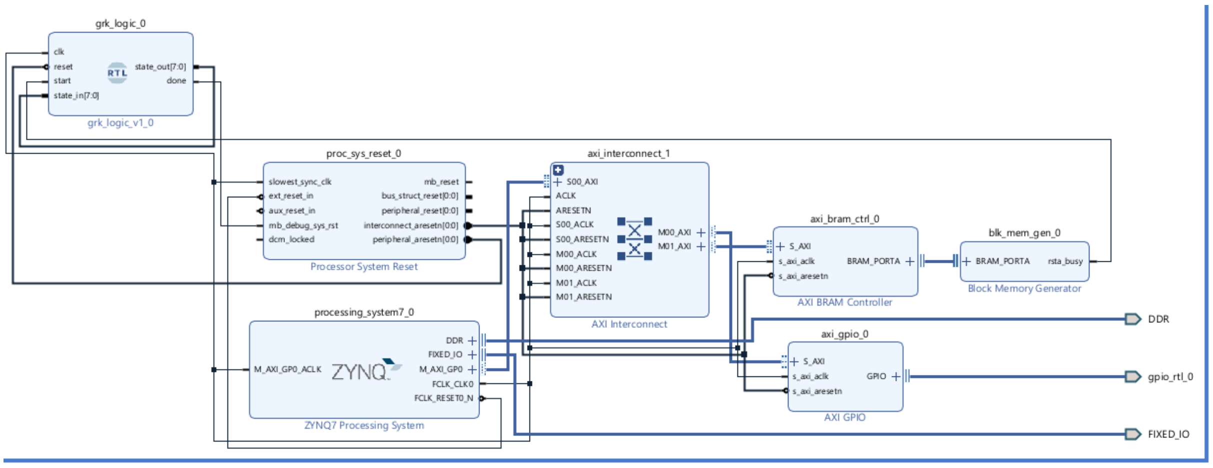
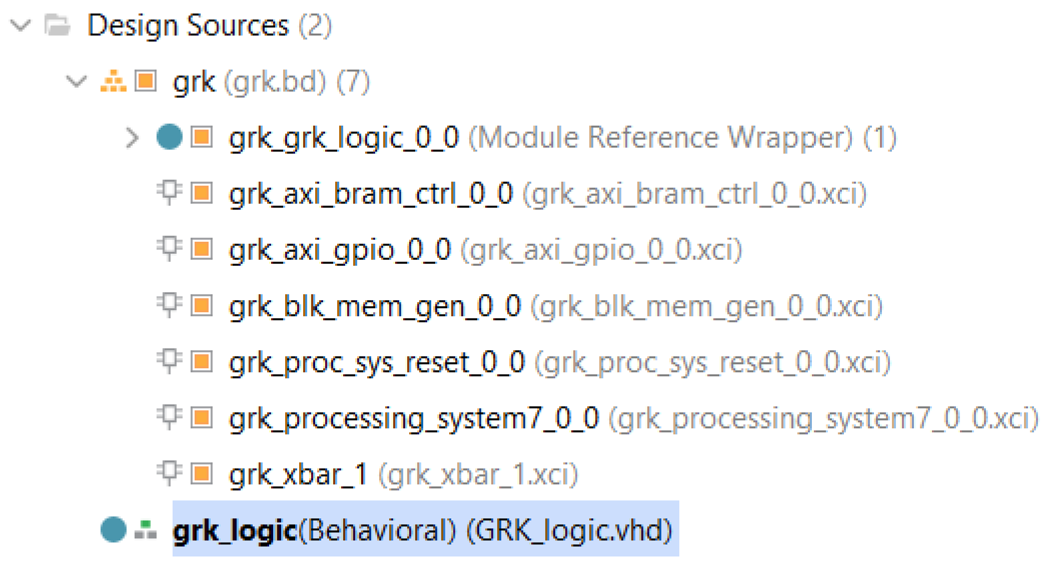


Fig 2. Block diagram of GRK algorithm using Zync7000(ZedBoard).

Implementing this algorithm on platforms like FPGA involves designing of block diagram structures such that we can reduce the number of oracle queries for reducing computation cost and compexiety. The GRK algorithm minimizes computational resources. Key implementation challenges include integrating quantum-inspired logic within classical computational systems and optimizing for hardware constraints [5] [6]. Fig 2 shows a block diagram which can show the flow of implementation of the proposed algorithm implemented on Xilinx Vivado 2023.1.

The modules used to design the block diagram are given in fig 3. And the functionality of the components are:

- Processing System (Zynq 7000): The central controller. It handles computation, algorithm setup and execution flow.
- grk_logic: Implements the GRK algorithm steps.
- AXI Interconnect: Manages data flow between the processing system, grk_logic, memory, and GPIO.
- Memory (BRAM): Stores intermediate data, such as state vectors or lookup tables required during GRK operations.
- GPIO: Provides external interfacing or debugging signals for input database.



```
Design Sources (2)
  grk (grk.bd) (7)
    grk_grk_logic_0_0 (Module Reference Wrapper) (1)
    grk_axi_bram_ctrl_0_0 (grk_axi_bram_ctrl_0_0.xci)
    grk_axi_gpio_0_0 (grk_axi_gpio_0_0.xci)
    grk_blk_mem_gen_0_0 (grk_blk_mem_gen_0_0.xci)
    grk_proc_sys_reset_0_0 (grk_proc_sys_reset_0_0.xci)
    grk_processing_system7_0_0 (grk_processing_system7_0_0.xci)
    grk_xbar_1 (grk_xbar_1.xci)
  grk_logic(Behavioral) (GRK_logic.vhd)
```

Fig 3. Modules used for designing block diagram on Xilinx Vivado 2023.1

Conclusion:

We discussed about the Grovers search algorithm which is an oracle based quantum technique to carry put search operations for a target bit from target blocks in databases. The optimised version is used for this study which is called the Grover-Radhakrishnan-Korepin (GRK) search for multi target and partial search. The methodology was first discussed followed by the quantum circuit mechanism and then finally the FPGA implementation was analysed.

References:

[1] R. P. da Silva, "Oracle Quantum Algorithms: An Overview," *Department of Mathematics, Faculty of Science, University of Porto*, June 11, 2024.

[2] P. R. Giri and V. E. Korepin, "A review on quantum search algorithms," *Quantum Information Processing*, vol. 16, no. 315, Nov. 2017. DOI: 10.1007/s11128-017-1768-7.

[3] V. E. Korepin and Y. Xu, "Binary Quantum Search," *arXiv preprint*, arXiv:0705.0777 [quant-ph], May 6, 2007. [Revised August 30, 2021].

[4]"Quantum Partial Search for Uneven Distribution of Multiple Target Items"
K. Zhang and V. Korepin, "Quantum Partial Search for Uneven Distribution of Multiple Target Items," *arXiv preprint*, arXiv:1703.01552 [quant-ph], 2017.

[5]"Quantum Partial Search Algorithm with Smaller Oracles for Multiple Target Items"
Y. Li and S. Wu, "Quantum Partial Search Algorithm with Smaller Oracles for Multiple Target Items," *arXiv preprint*, arXiv:2206.05022 [quant-ph], 2022.

[6] V. E. Korepin, "Optimization of Partial Search," *C.N. Yang Institute for Theoretical Physics, State University of New York at Stony Brook*, Stony Brook, NY, 2005. Available at: arXiv:quant-ph/0503238.

Appendix:

a)Gkr_ckt (quantum circuit code)

```
from qiskit import QuantumCircuit

from qiskit.visualization import plot_histogram, circuit_drawer

from qiskit_aer import AerSimulator

import numpy as np

import matplotlib.pyplot as plt


def grk_oracle(n_qubits, marked_states):

   oracle = QuantumCircuit(n_qubits)

   for state in marked_states:

      for i, bit in enumerate(state):

         if bit == '0':

            oracle.x(i)

      oracle.h(n_qubits - 1)

      oracle.mcx(list(range(n_qubits - 1)), n_qubits - 1)
```

```
        oracle.h(n_qubits - 1)
        for i, bit in enumerate(state):
            if bit == '0':
                oracle.x(i)

    return oracle

def grk_diffusion_operator(n_qubits):
    diffusion = QuantumCircuit(n_qubits)
    diffusion.h(range(n_qubits - 1))
    diffusion.x(range(n_qubits - 1))
    diffusion.h(n_qubits - 1)
    diffusion.mcx(list(range(n_qubits - 1)), n_qubits - 1)
    diffusion.h(n_qubits - 1)
    diffusion.x(range(n_qubits - 1))
    diffusion.h(range(n_qubits - 1))

    return diffusion

n_qubits = 4
ancilla_qubit = n_qubits - 1
marked_states = ['011', '101']
grk_circuit = QuantumCircuit(n_qubits, n_qubits - 1)
grk_circuit.h(range(n_qubits - 1))
oracle = grk_oracle(n_qubits, marked_states)
grk_circuit.compose(oracle, inplace=True)
diffusion = grk_diffusion_operator(n_qubits)
grk_circuit.compose(diffusion, inplace=True)
n_iterations = int(np.sqrt((2 ** (n_qubits - 1)) / len(marked_states)))
```

```
for _ in range(n_iterations - 1):
    grk_circuit.compose(oracle, inplace=True)
    grk_circuit.compose(diffusion, inplace=True)
grk_circuit.measure(range(n_qubits - 1), range(n_qubits - 1))
print("GRK Quantum Circuit:")
print(grk_circuit)
grk_circuit.draw(output='mpl')
plt.show()
```

b)Grk_logic(logical block diagram VHDL module code)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity grk_logic is
    Port (
        clk       : in  std_logic;
        reset     : in  std_logic;
        start     : in  std_logic;
        state_in  : in  std_logic_vector(7 downto 0);
        state_out : out std_logic_vector(7 downto 0);
        done      : out std_logic
    );
end grk_logic;

architecture Behavioral of grk_logic is
    -- Internal registers
    signal marked_state      : std_logic_vector(7 downto 0) := "10101010"; -- Marked state
    signal iteration_counter : std_logic_vector(3 downto 0) := (others => '0');
    signal state_out_reg     : std_logic_vector(7 downto 0);
    signal done_reg          : std_logic;

begin
    process(clk, reset)
    begin
        if reset = '1' then
            state_out_reg <= (others => '0');
            done_reg <= '0';
            iteration_counter <= (others => '0');

        elsif rising_edge(clk) then
            if start = '1' then
                if state_in = marked_state then
                    state_out_reg <= state_in; -- Marked state found
                    done_reg <= '1';
                else
                    iteration_counter <= iteration_counter + '1';
                    if iteration_counter = "0100" then -- Example: Stop after 4 iterations
                        done_reg <= '1';
                    end if;
                end if;
            end if;
        end if;
    end process;

    -- Assign internal registers to outputs
    state_out <= state_out_reg;
    done <= done_reg;

end Behavioral;
```