Proposal for an Open Source Flash Failure Analysis Platform (FLAP)

By Michael Tomer, Cory Shirts, SzeHsiang Harper, Jake Johns

http://code.google.com/p/uofu2009-2010clinicteam/

Introduction

- Cory

Introduction

- Flash Memory prevalence
  - Cell Phones, MP3 Players, Cameras, Hard Drives
- Still a new Technology
  - NAND flash memory has a limited number of read/write cycles, its behavior past this limit has not been widely analyzed
- Goals
  - Create a open source system to test NAND flash memory

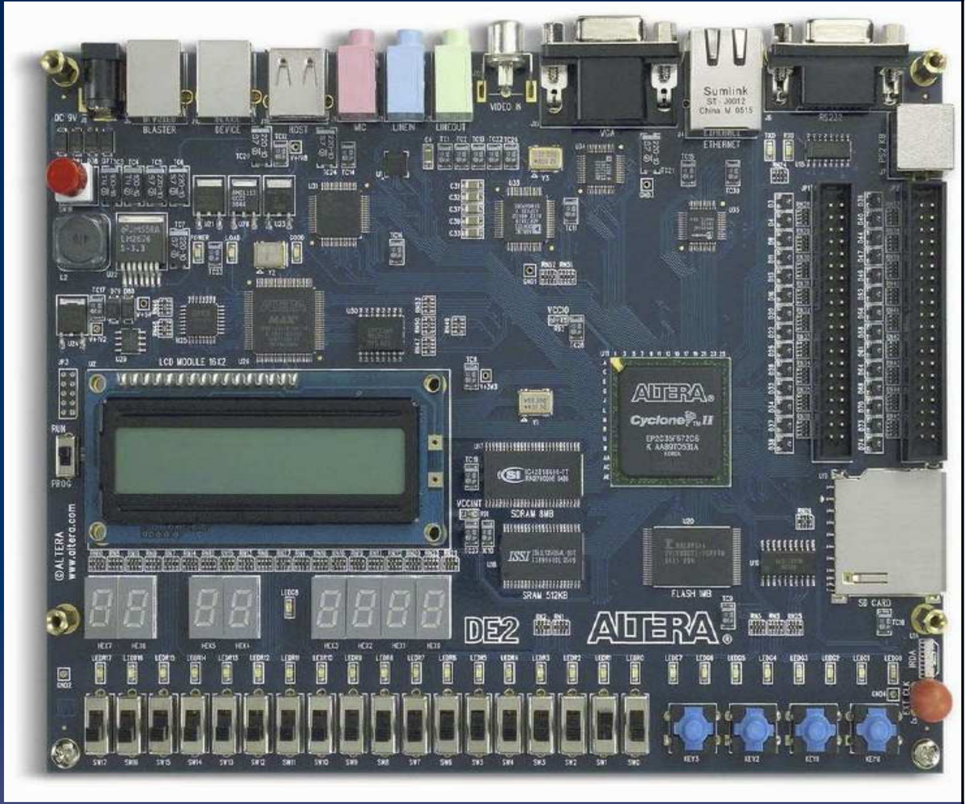
Bill of Materials

- Provided through the University
  - Altera-DE2 Development & Education Board
  - USB Cables
- Provided by Micron
  - NAND Flash storage
  - NAND Flash Daughter Board

Bill of Materials

- Software (Free Downloads)
  - LibUSBDotNet (SourceForge)
  - Visual Studio Express (C# version)
  - Altera Quartus II Web Edition Verilog dev. environment
  - Altera Nios II Embedded Design Suite

Daughter Board / Memory Controller and FPGA



- Jake

FPGA

- Altera DE2 Development Board

  Includes:

  – System On a Programmable Chip (SOPC)

  – NAND Controller

  – Clock Generator

  – Reset De-Bouncer

  – On-chip dual port RAM

  – Parts integrated using Verilog

NIOS 2 Embedded processor

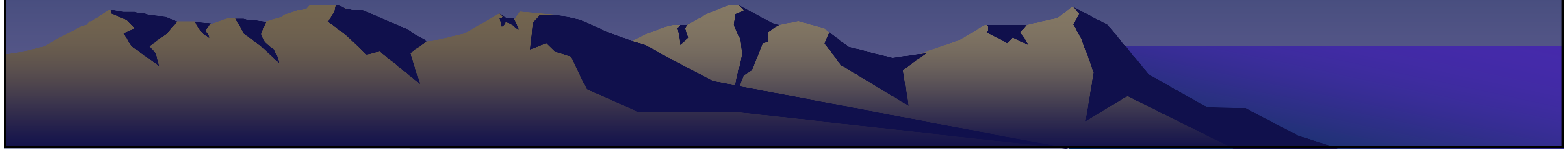
- Programmed with C
- Controls Interfaces
  - USB
  - To the GUI on the computer
  - To the Daughtboard
- Controls Displays
- Stores test results.

NAND Controller

- Direct Interface for controlling the NAND flash
- Runs with 66 MHz clock.
- Deals with the commands:
  - Read
  - Program
  - Erase
  - Read ID
  - Reset
  - Read Status

On-chip port RAM

- Used as a buffer
  - Receive data
  - Sending commands
- Controlled by two signals
- Used by:
  - NIOS 2
  - NAND Controller



Reset De-Bouncer

- Hardware reset
- Debounces reset
- Waits for the clocks to be valid

USB interface

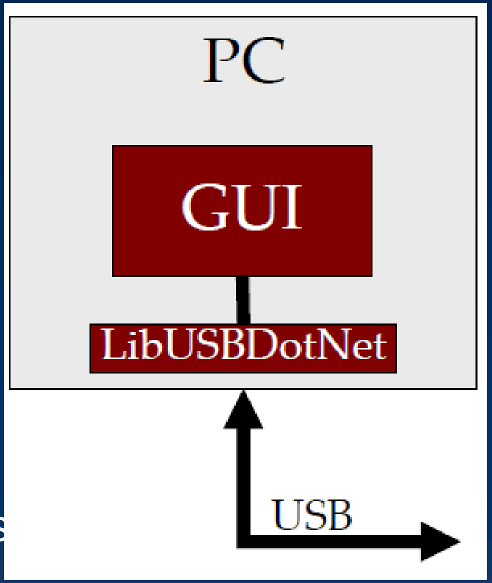
- Sze

USB interface

- FPGA USB interface
  - Communicates with the host PC
  - Is programmed in firmware
  - Responsible for:
    - receiving commands from the host PC
    - Transmitting results back to the host PC

GUI → USB

- Uses the LibUSBDotNet C# libraries to instantiate the device and communicate over the USB endpoints
  - –Don’t have to write a Windows driver!
  - –Driver runs using managed code in user space
- 32 byte command sent from GUI to firmware
  - 0: opcode
  - 1: seed
  - 2: algorithm
  - 3: debug level
  - 4-7: cycles
  - 8-11: start address
    - { 00, block(12), page(6), column(12) } = 32 bits
  - 12-15: end address
    - { 00, block(12), page(6), column(12) } = 32 bits
  - 16-31: reserved

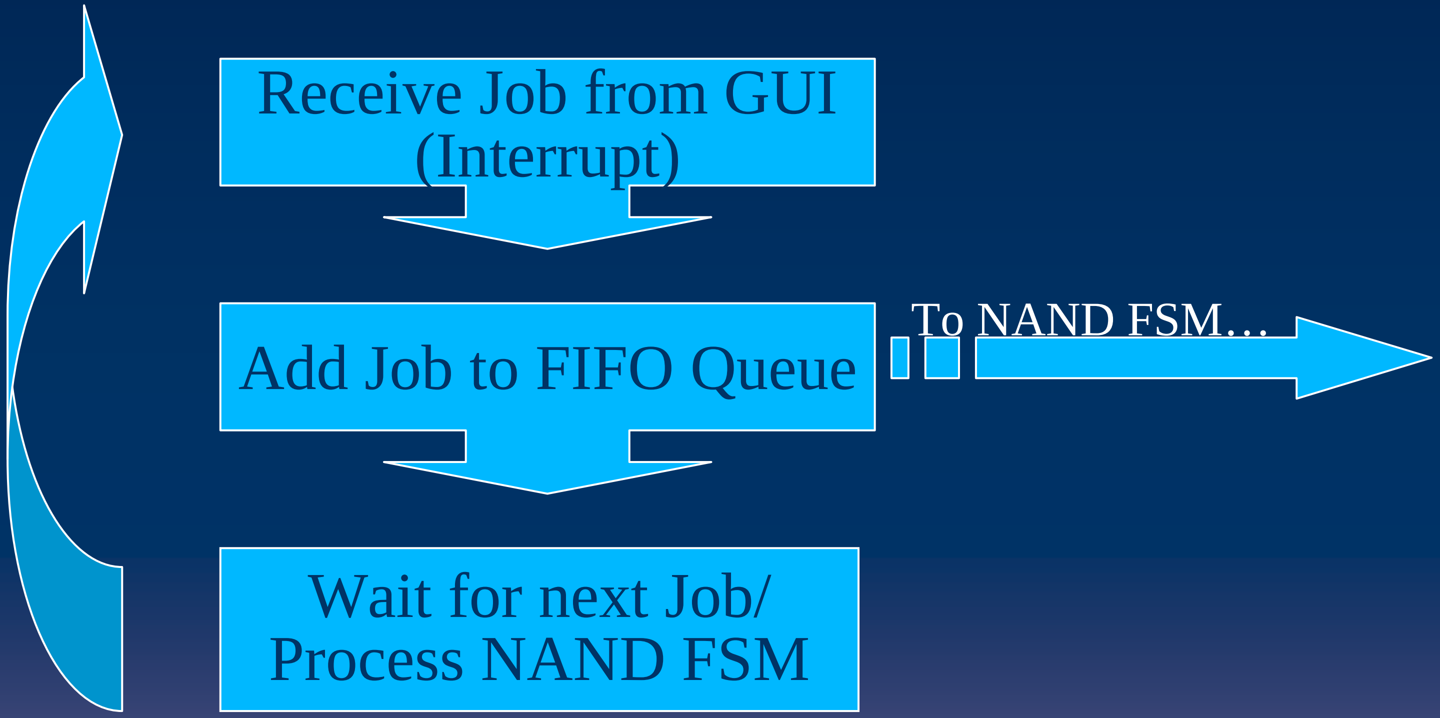


USB → GUI

- Debug Endpoint
  - Sends information about firmware state according to the “debug” level sent in command
- Status Endpoint
  - Returns periodic status information about the progress of the job

USB → Firmware

State Machine for USB portion of firmware:



Receive Job from GUI (Interrupt)

Add Job to FIFO Queue

Wait for next Job/ Process NAND FSM

To NAND FSM…

USB → GUI

- Result Endpoint
  - Sends the results of jobs as XML data

```
<job id="4" opcode="128" seed="0" algorithm="0" cycles="10000"
    startAddress="0x00000000" endAddress="0x000FC000" debug="0">
    <data>U3VjayBteSBiYWxscyE=</data>
    <error count="1" address="0x0x000FC000">
      <byte index="23" received="35"/>
      <byte index="444" received="255"/>
    </error>
    <time days="0" hours="0" minutes="0" seconds="7" millisec="519"/>
    <done failureCode="0" failures="1"/>
</job>
```

USB interface

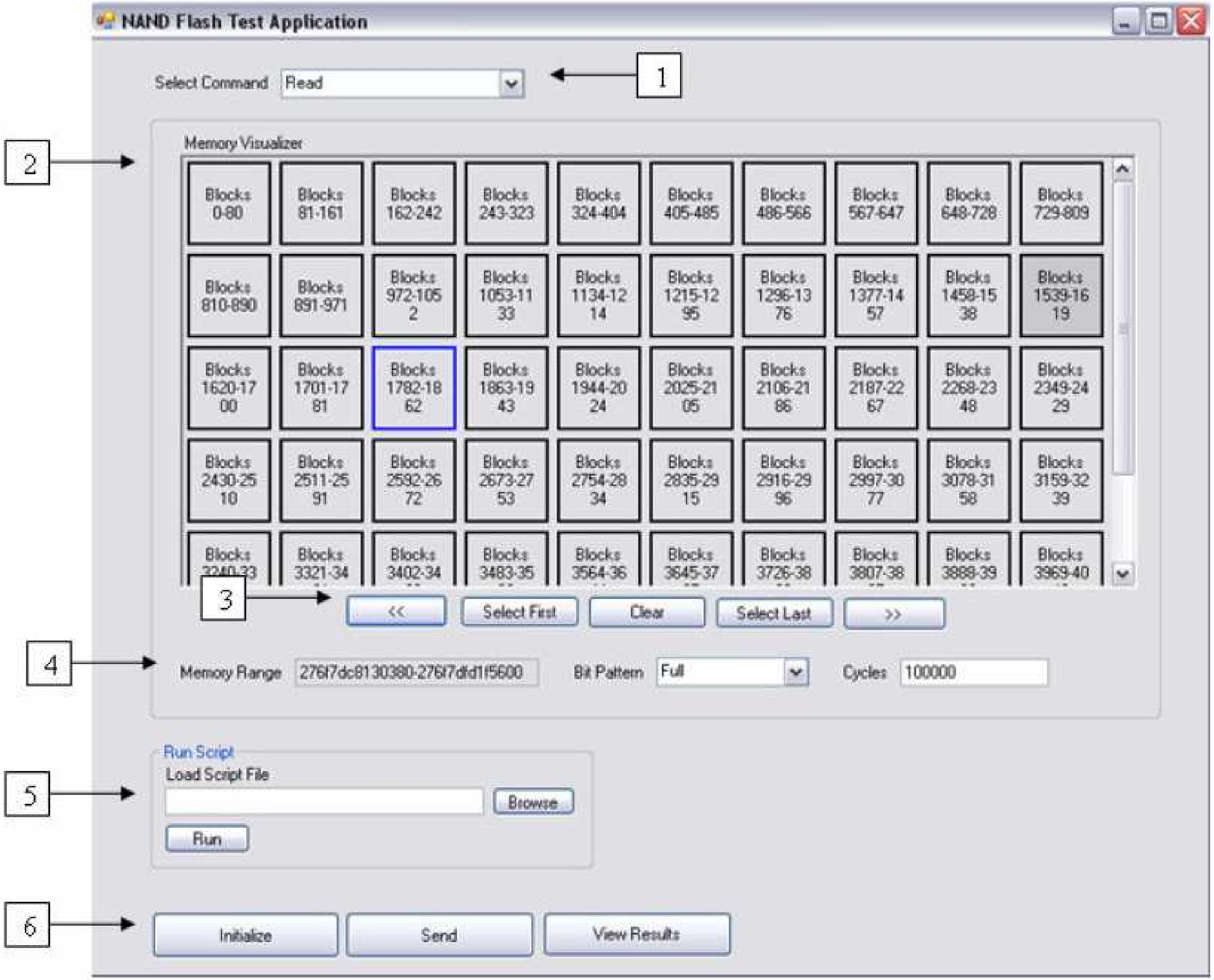
- USB interface on host PC stores results in a SQL database
  - using the ActiveX Data Objects Classes of the .NET framework to communicate with the database
    - Normalized database
    - T-SQL (Transactional SQL)
      - extension to the SQL database programming language
    - Initially SQL database will store basic info but can be expanded

Graphical User Interface on the PC

- Mike

GUI

- 2006-2007 Team completed a very basic GUI
- They were unable to fully test it because of the problems with the USB



Select Command: Read

Blocks 0-80	Blocks 81-161	Blocks 162-242	Blocks 243-323	Blocks 324-404	Blocks 405-485	Blocks 486-566	Blocks 567-647	Blocks 648-728	Blocks 729-809
Blocks 810-890	Blocks 891-971	Blocks 972-1052	Blocks 1053-1133	Blocks 1134-1214	Blocks 1215-1295	Blocks 1296-1376	Blocks 1377-1457	Blocks 1458-1538	Blocks 1539-1619
Blocks 1620-1700	Blocks 1701-1781	Blocks 1782-1862	Blocks 1863-1943	Blocks 1944-2024	Blocks 2025-2105	Blocks 2106-2186	Blocks 2187-2267	Blocks 2268-2348	Blocks 2349-2429
Blocks 2430-2510	Blocks 2511-2591	Blocks 2592-2672	Blocks 2673-2753	Blocks 2754-2834	Blocks 2835-2915	Blocks 2916-2996	Blocks 2997-3077	Blocks 3078-3158	Blocks 3159-3239
Blocks 3240-33	Blocks 3321-34	Blocks 3402-34	Blocks 3483-35	Blocks 3564-36	Blocks 3645-37	Blocks 3726-38	Blocks 3807-38	Blocks 3888-39	Blocks 3969-40

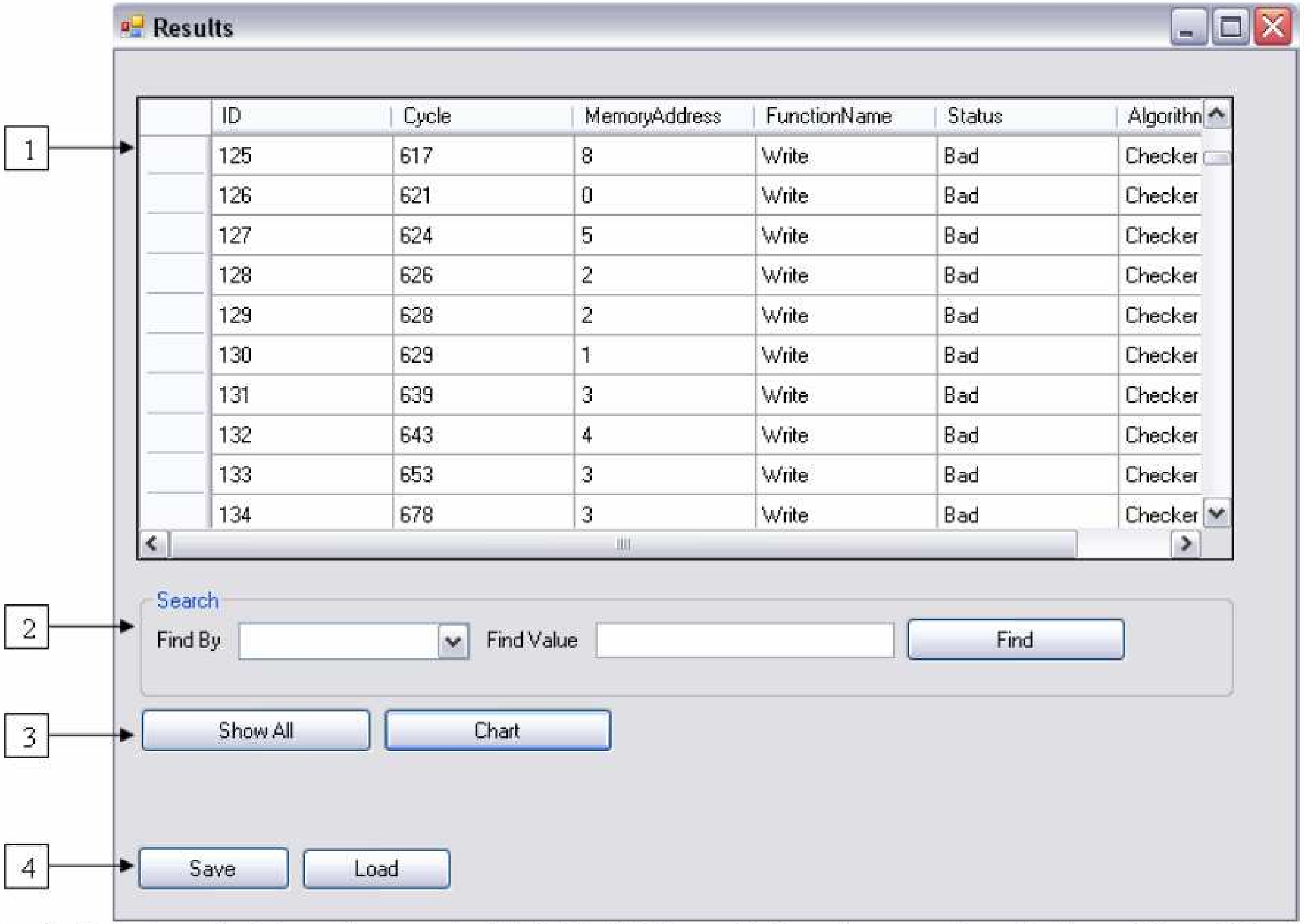
Memory Range: 276f7dc8130380-276f7dfd1f5600

Bit Pattern: Full

Cycles: 100000

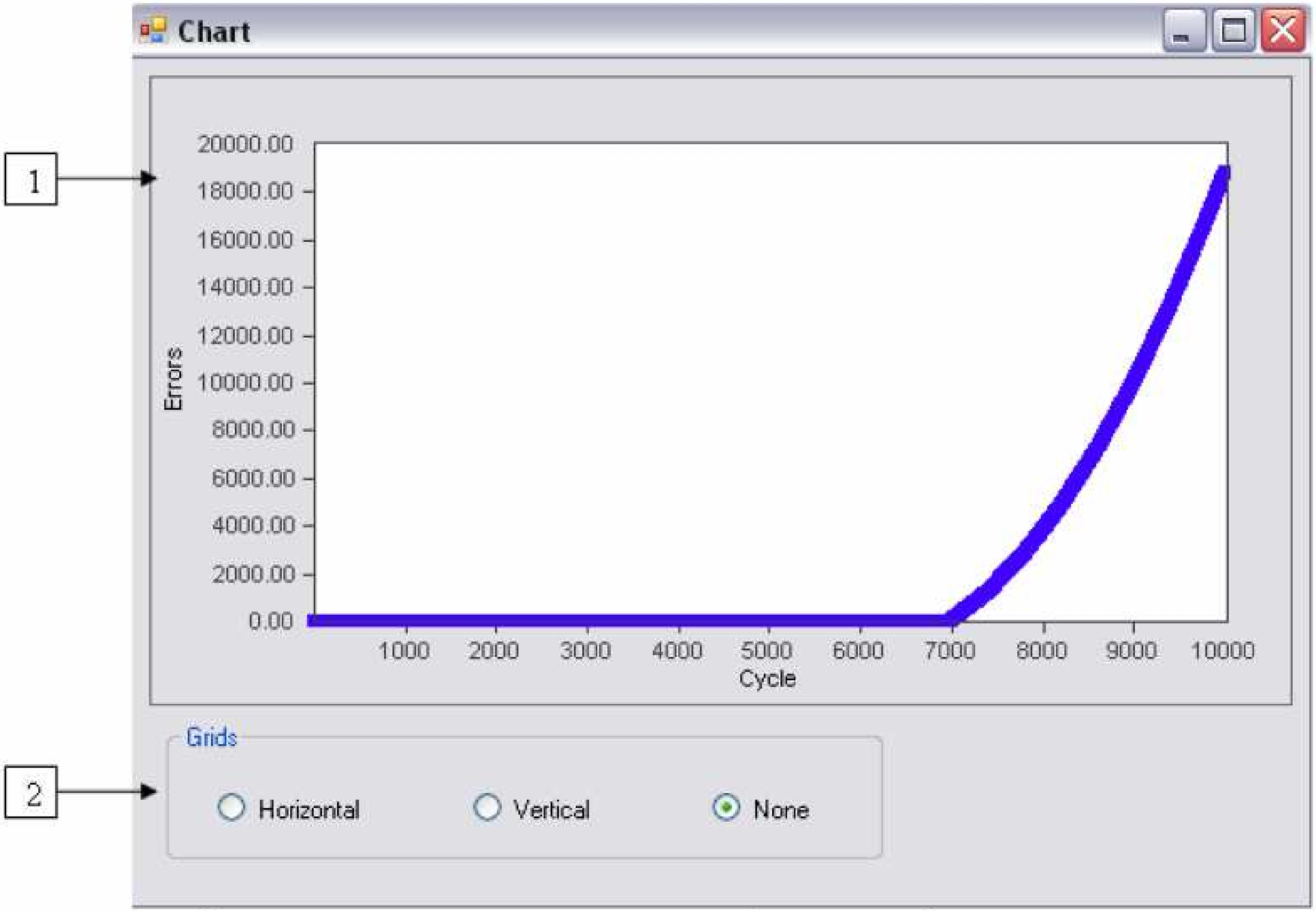
Load Script File

Figure 2: Command Interface 1. Select Command combo box 2. Memory Visualizer 3. Navigation and memory range selection buttons 4. Memory Range, Bit Pattern, and Cycles text boxes 5. Script group box 6. Initialize, Send, and View Results buttons



ID	Cycle	MemoryAddress	FunctionName	Status	Algorithn
125	617	8	Write	Bad	Checker
126	621	0	Write	Bad	Checker
127	624	5	Write	Bad	Checker
128	626	2	Write	Bad	Checker
129	628	2	Write	Bad	Checker
130	629	1	Write	Bad	Checker
131	639	3	Write	Bad	Checker
132	643	4	Write	Bad	Checker
133	653	3	Write	Bad	Checker
134	678	3	Write	Bad	Checker

Figure 3: Results Interface 1. Data set connected to SQL database 2. Search group box 3. Show All and Chart buttons 4. Save and Load buttons



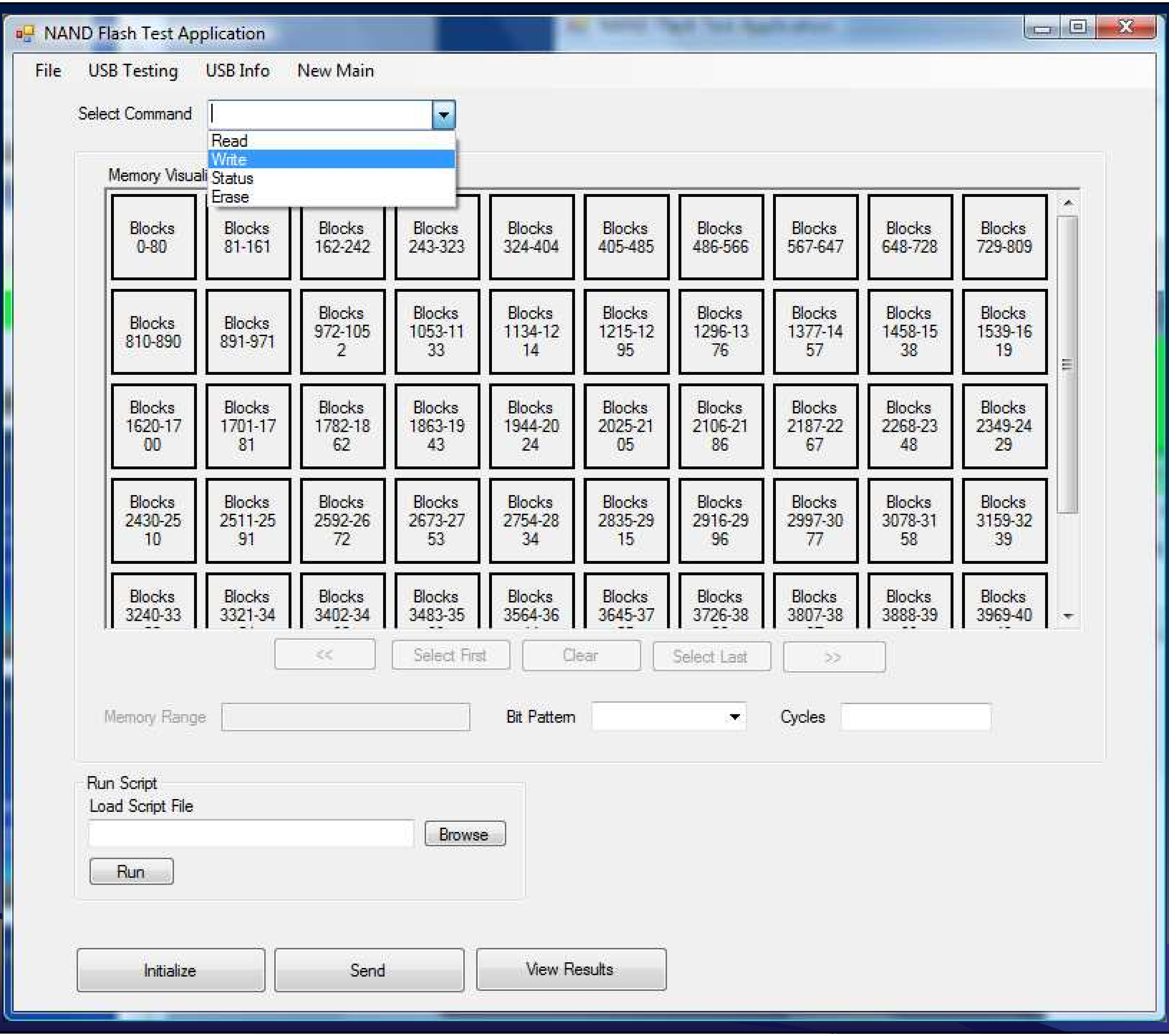
Grids

○ Horizontal ○ Vertical ● None

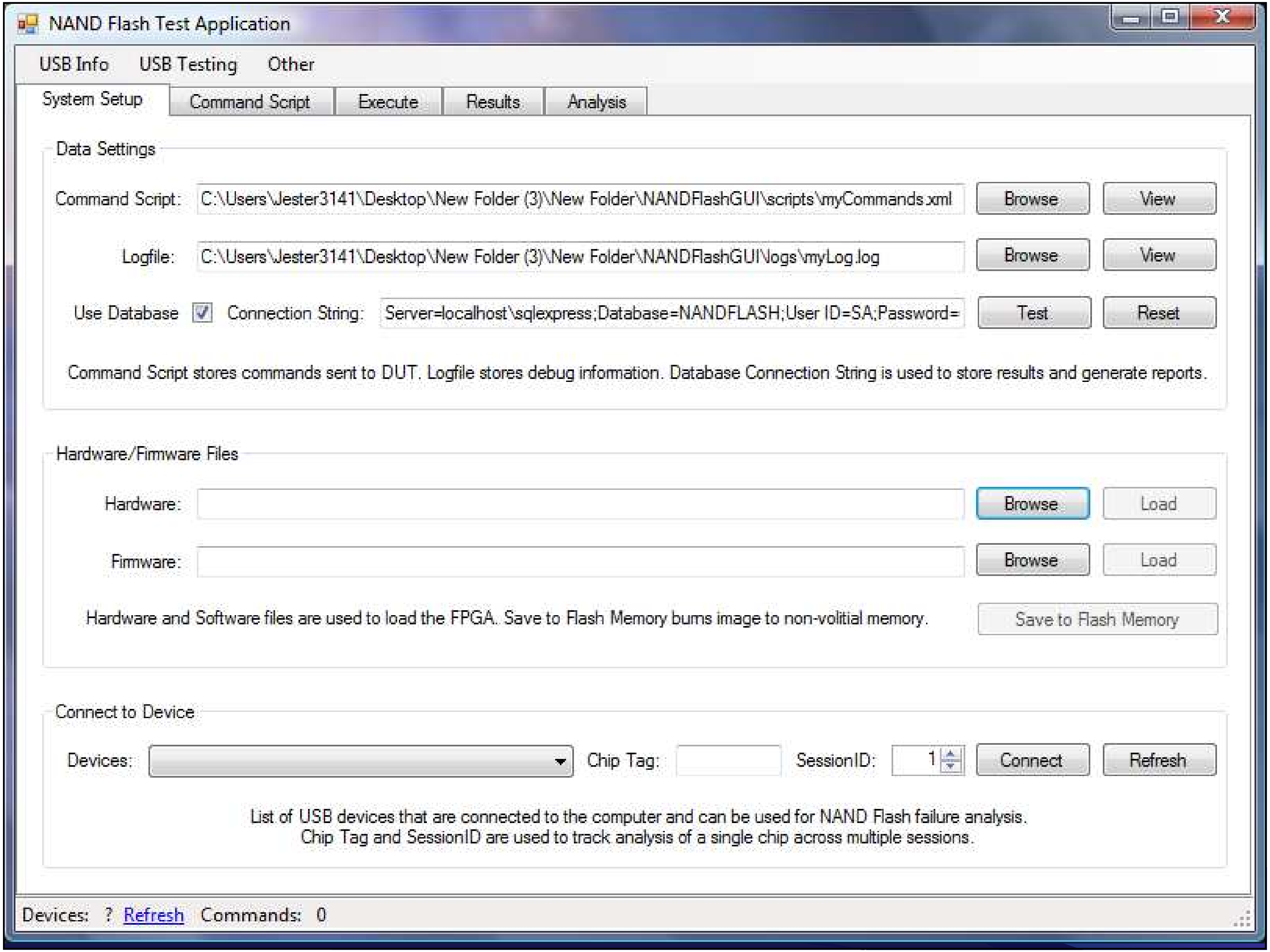
Figure 4: Results Interface 1. Chart display 2. Grids group box

Graphical User Interface on the PC

- Need to expand on 2006-2007’s interface
  - Automated Testing patterns can be specified
    - range of blocks to test
    - Number of cycles to run for
    - Can use specific memory patters or randomly generated patterns for testing.
  - Options for connecting to the database
  - Loading the firmware onto the FPGA



Blocks 0-80	Blocks 81-161	Blocks 162-242	Blocks 243-323	Blocks 324-404	Blocks 405-485	Blocks 486-566	Blocks 567-647	Blocks 648-728	Blocks 729-809
Blocks 810-890	Blocks 891-971	Blocks 972-105 2	Blocks 1053-11 33	Blocks 1134-12 14	Blocks 1215-12 95	Blocks 1296-13 76	Blocks 1377-14 57	Blocks 1458-15 38	Blocks 1539-16 19
Blocks 1620-17 00	Blocks 1701-17 81	Blocks 1782-18 62	Blocks 1863-19 43	Blocks 1944-20 24	Blocks 2025-21 05	Blocks 2106-21 86	Blocks 2187-22 67	Blocks 2268-23 48	Blocks 2349-24 29
Blocks 2430-25 10	Blocks 2511-25 91	Blocks 2592-26 72	Blocks 2673-27 53	Blocks 2754-28 34	Blocks 2835-29 15	Blocks 2916-29 96	Blocks 2997-30 77	Blocks 3078-31 58	Blocks 3159-32 39
Blocks 3240-33	Blocks 3321-34	Blocks 3402-34	Blocks 3483-35	Blocks 3564-36	Blocks 3645-37	Blocks 3726-38	Blocks 3807-38	Blocks 3888-39	Blocks 3969-40



Data Settings

Command Script: C:\Users\Jester3141\Desktop\New Folder (3)\New Folder\NANDFlashGUI\scripts\myCommands.xml

Logfile: C:\Users\Jester3141\Desktop\New Folder (3)\New Folder\NANDFlashGUI\logs\myLog.log

Use Database ☑ Connection String: Server=localhost\sqlexpress;Database=NANDFLASH;User ID=SA;Password=

Command Script stores commands sent to DUT. Logfile stores debug infomation. Database Connection String is used to store results and generate reports.

Hardware/Firmware Files

Hardware:

Firmware:

Hardware and Software files are used to load the FPGA. Save to Flash Memory burns image to non-volitial memory.

Connect to Device

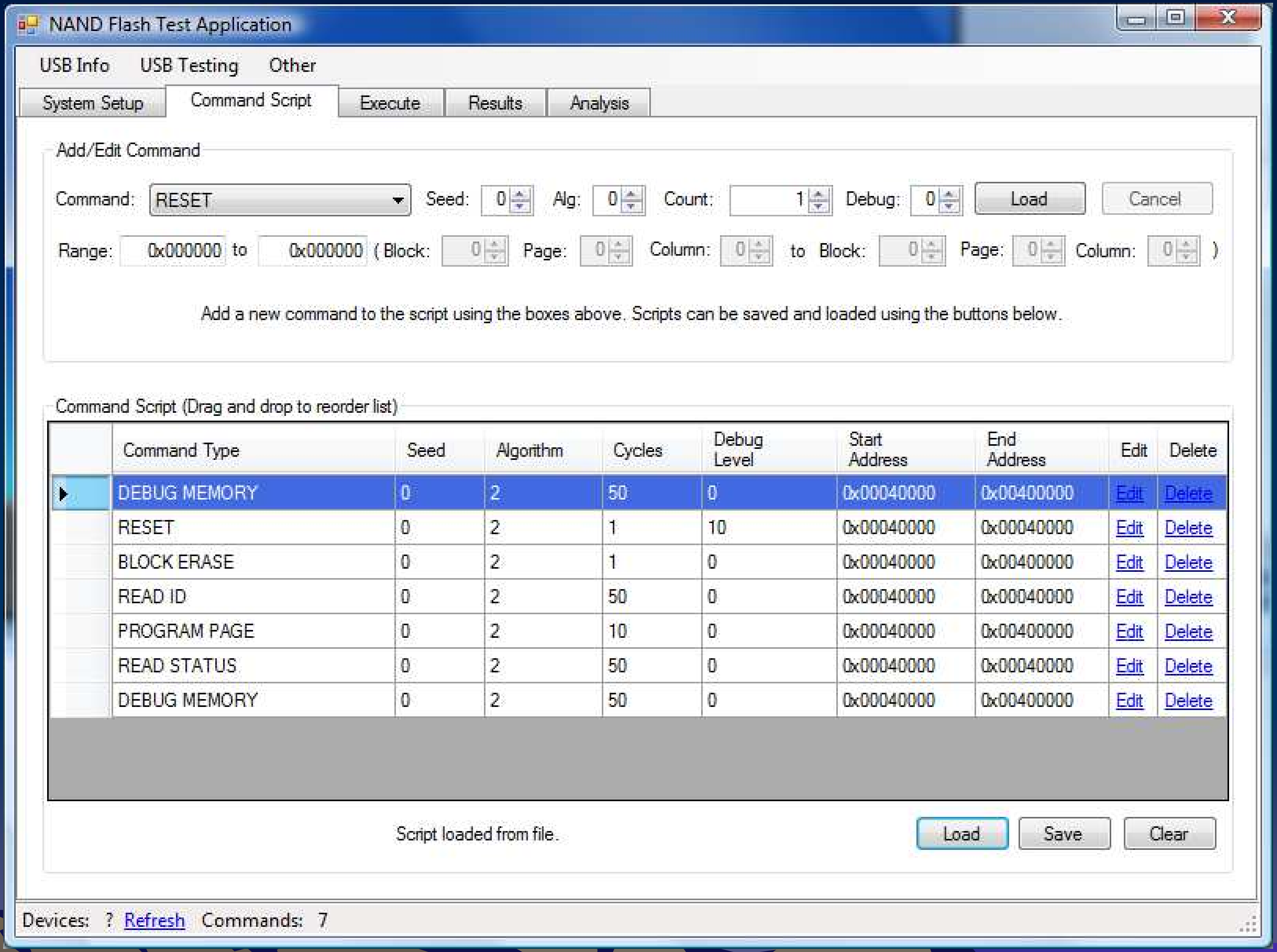
Devices:

Chip Tag:

SessionID: 1

List of USB devices that are connected to the computer and can be used for NAND Flash failure analysis.
Chip Tag and SessionID are used to track analysis of a single chip across multiple sessions.

Devices: ? Refresh Commands: 0



Command: RESET Seed: 0 Alg: 0 Count: 1 Debug: 0

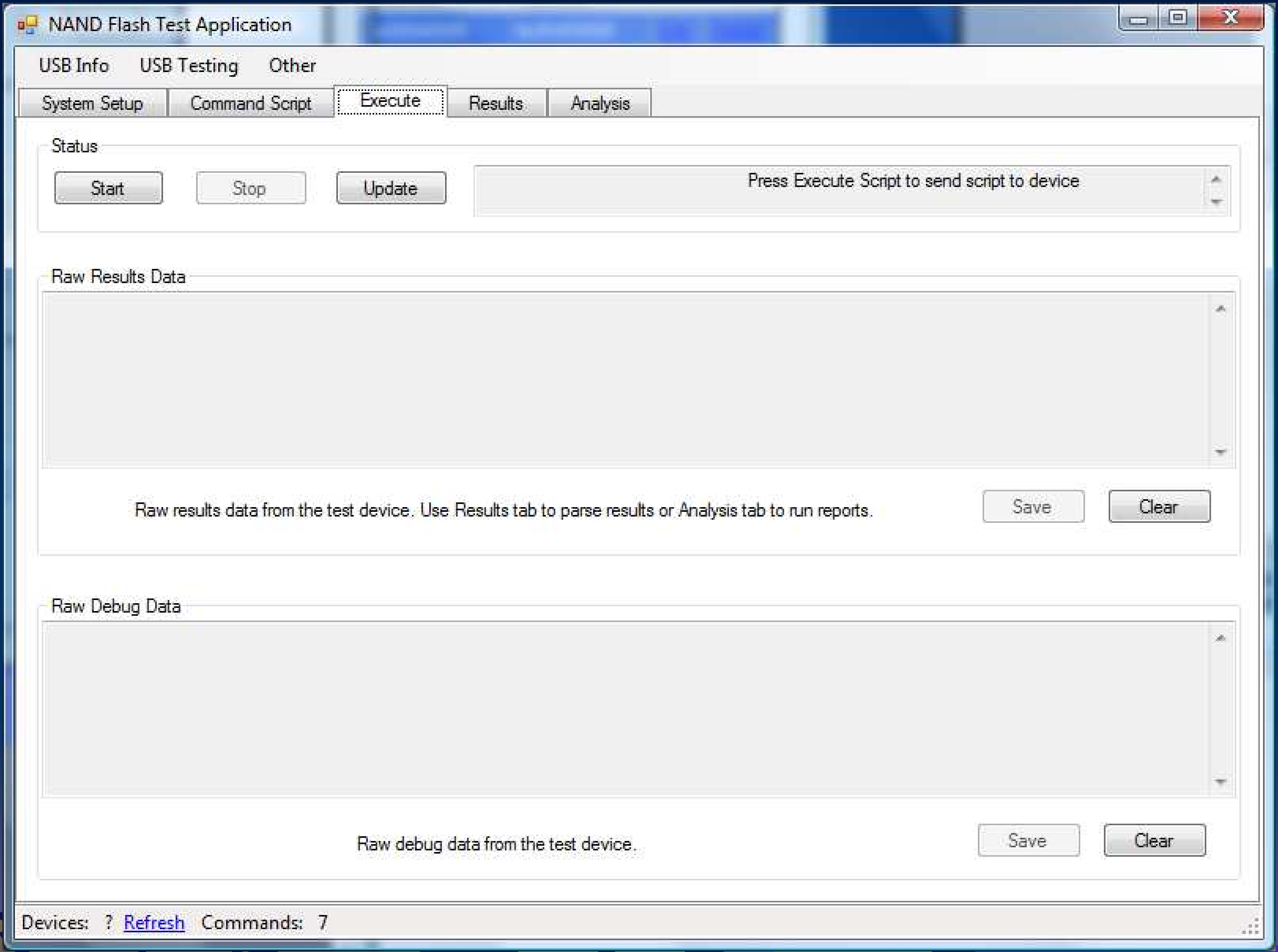
Range: 0x000000 to 0x000000 (Block: 0 Page: 0 Column: 0 to Block: 0 Page: 0 Column: 0)

Add a new command to the script using the boxes above. Scripts can be saved and loaded using the buttons below.

Command Type	Seed	Algorithm	Cycles	Debug Level	Start Address	End Address	Edit	Delete
DEBUG MEMORY	0	2	50	0	0x00040000	0x00400000	Edit	Delete
RESET	0	2	1	10	0x00040000	0x00040000	Edit	Delete
BLOCK ERASE	0	2	1	0	0x00040000	0x00400000	Edit	Delete
READ ID	0	2	50	0	0x00040000	0x00040000	Edit	Delete
PROGRAM PAGE	0	2	10	0	0x00040000	0x00400000	Edit	Delete
READ STATUS	0	2	50	0	0x00040000	0x00040000	Edit	Delete
DEBUG MEMORY	0	2	50	0	0x00040000	0x00400000	Edit	Delete

Script loaded from file.

Devices: ? Refresh Commands: 7

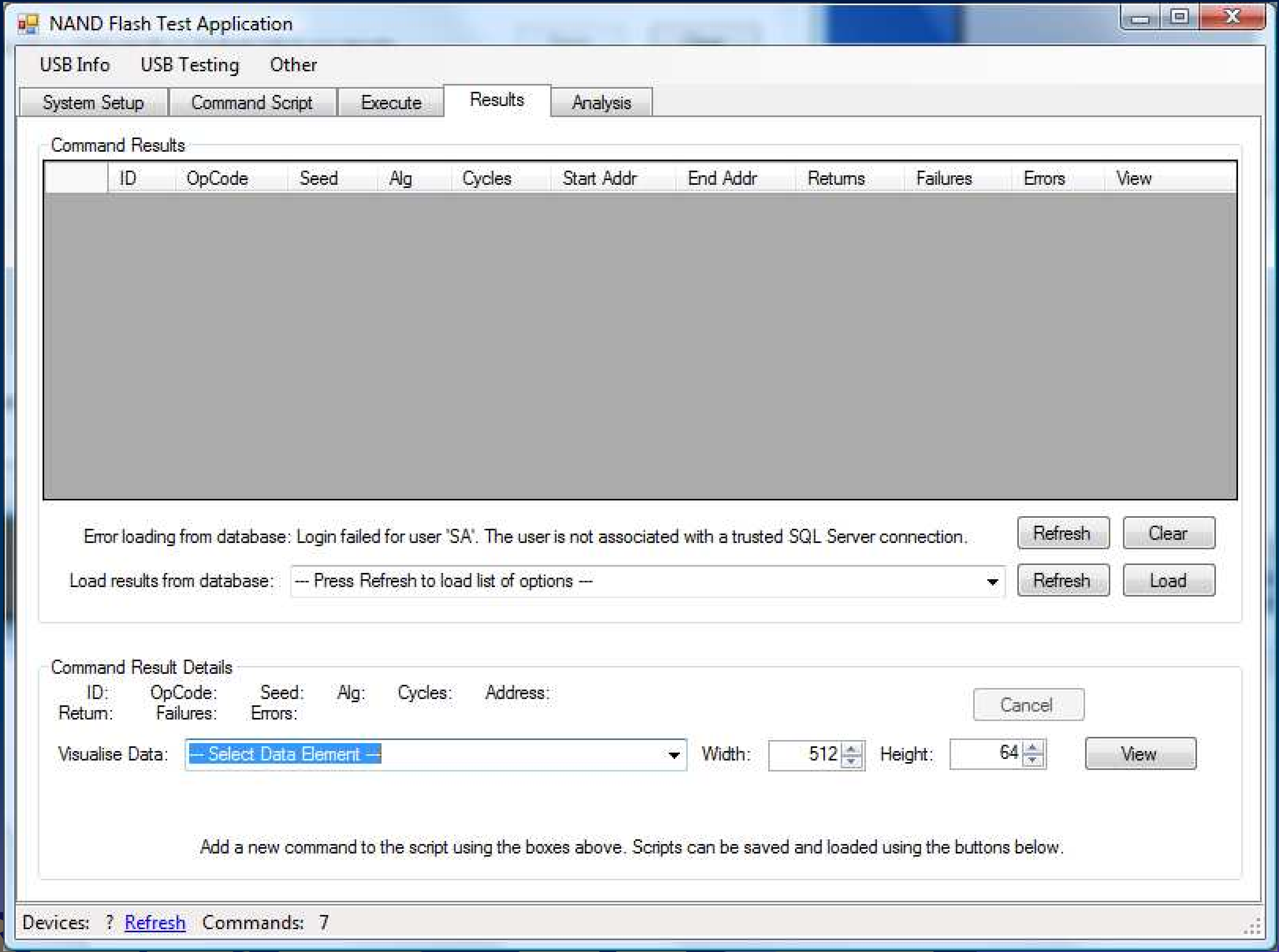


Press Execute Script to send script to device

Raw results data from the test device. Use Results tab to parse results or Analysis tab to run reports.

Raw debug data from the test device.

Devices: ? Refresh Commands: 7



ID	OpCode	Seed	Alg	Cycles	Start Addr	End Addr	Returns	Failures	Errors	View

Error loading from database: Login failed for user 'SA'. The user is not associated with a trusted SQL Server connection.

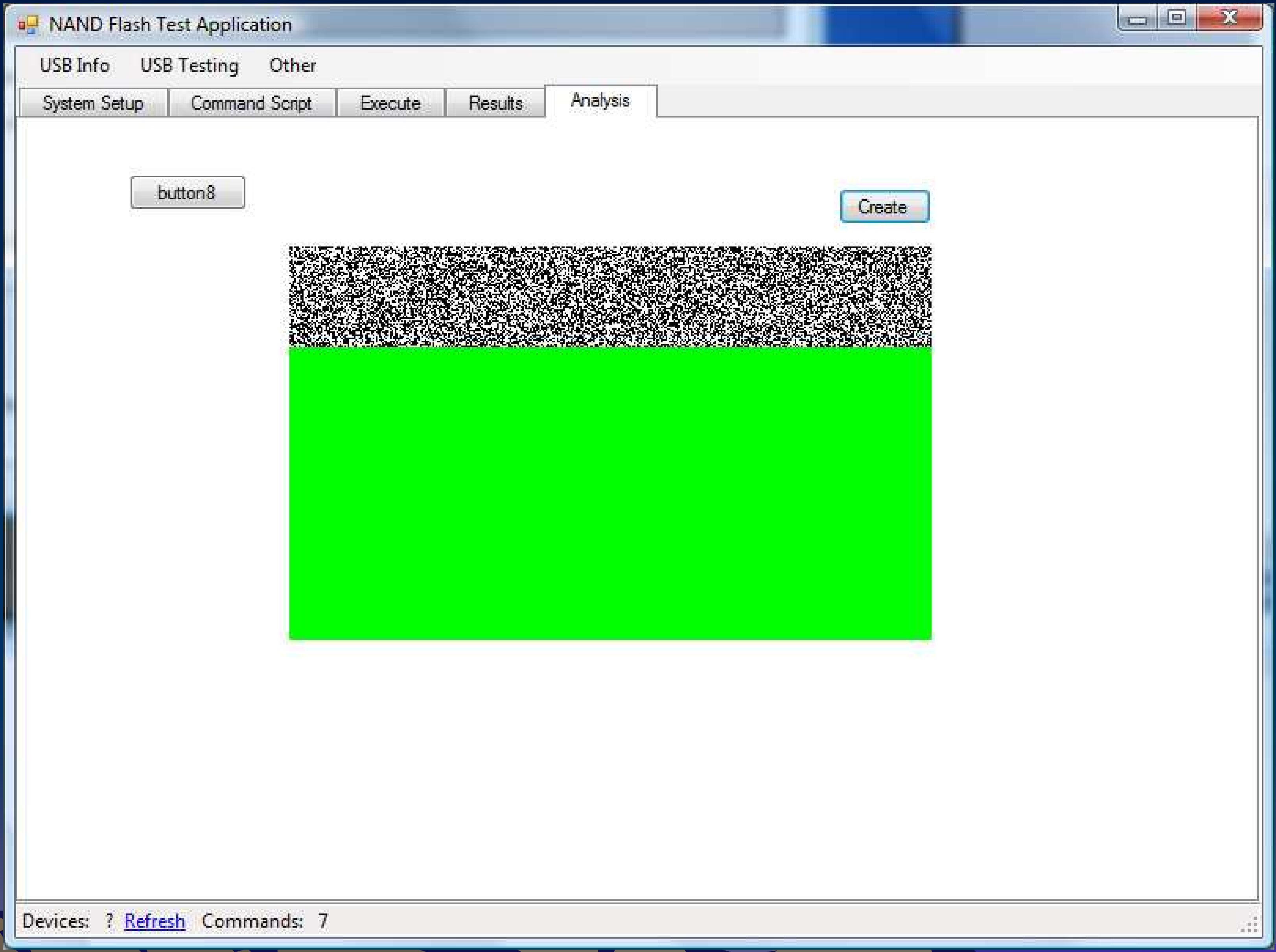
Load results from database: --- Press Refresh to load list of options ---

ID: OpCode: Seed: Alg: Cycles: Address:
Return: Failures: Errors:

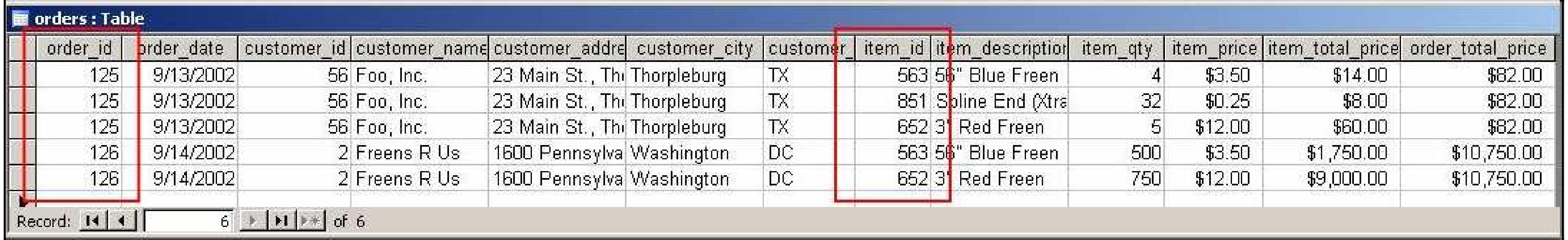
Visualise Data: -- Select Data Element -- Width: 512 Height: 64

Add a new command to the script using the boxes above. Scripts can be saved and loaded using the buttons below.

Devices: ? Refresh Commands: 7


NAND Flash Test Application
USB Info
USB Testing
Other
System Setup
Command Script
Execute
Results
Analysis
button8
Create
Devices: ? Refresh Commands: 7

Database Structure

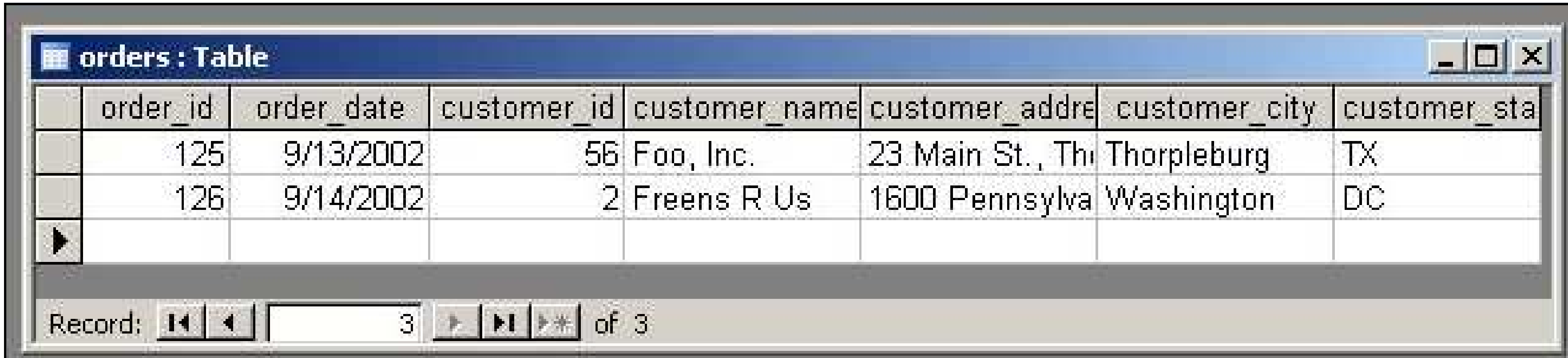


orders : Table

order_id	order_date	customer_id	customer_name	customer_addre	customer_city	customer_	item_id	item_descriptio	item_qty	item_price	item_total_price	order_total_price
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX	563	56" Blue Freen	4	$3.50	$14.00	$82.00
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX	851	Spline End (Xtra	32	$0.25	$8.00	$82.00
125	9/13/2002	56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX	652	3' Red Freen	5	$12.00	$60.00	$82.00
126	9/14/2002	2	Freens R Us	1600 Pennsylva	Washington	DC	563	56" Blue Freen	500	$3.50	$1,750.00	$10,750.00
126	9/14/2002	2	Freens R Us	1600 Pennsylva	Washington	DC	652	3' Red Freen	750	$12.00	$9,000.00	$10,750.00

Record: 6 of 6

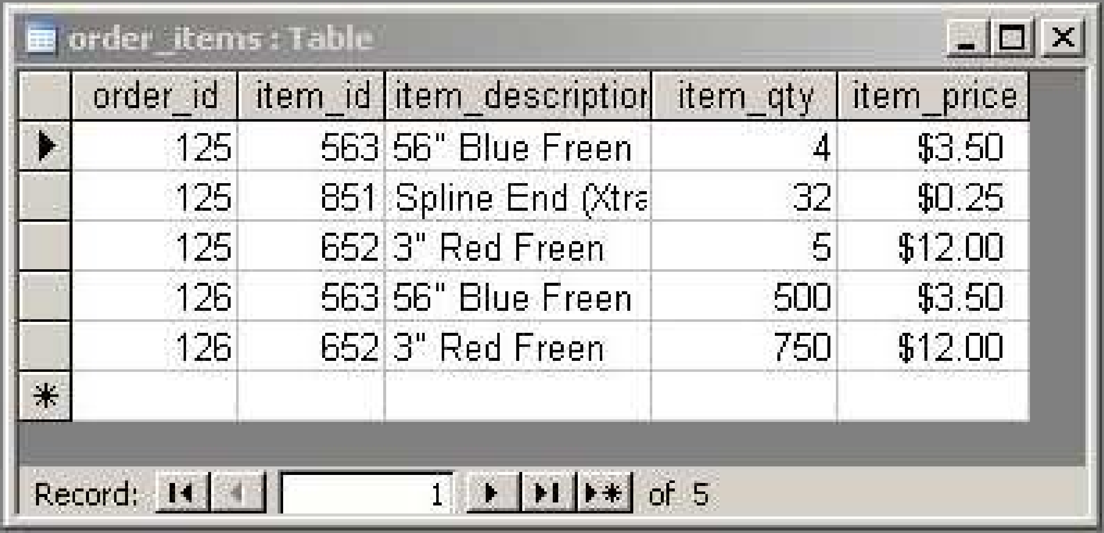
- Importance of data normalization
  - Non-Normalized data
    - Data is duplicated across many items
    - This leads to problems when updating and querying data and adding additional fields
      - If there is time we are planning to store additional information in the database
    - Uses more space
      - Space is critical because of the amount of data being stored.



orders : Table

order_id	order_date	customer_id	customer_name	customer_addre	customer_city	customer_sta
125	9/13/2002	56	Foo, Inc.	23 Main St., Thi	Thorpleburg	TX
126	9/14/2002	2	Freens R Us	1600 Pennsylva	Washington	DC

Record: 3 of 3

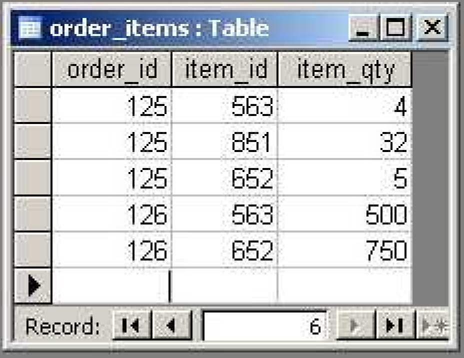


order_items : Table

order_id	item_id	item_descriptior	item_qty	item_price
125	563	56" Blue Freen	4	$3.50
125	851	Spline End (Xtra	32	$0.25
125	652	3" Red Freen	5	$12.00
126	563	56" Blue Freen	500	$3.50
126	652	3" Red Freen	750	$12.00

Record: 1 of 5

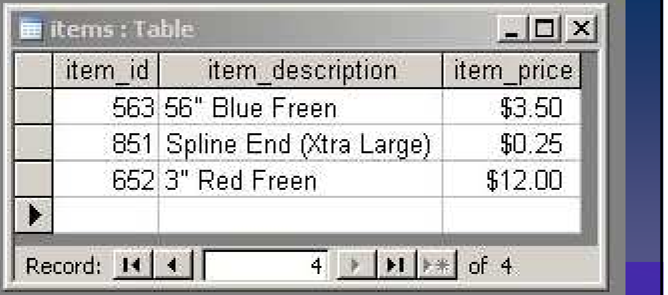
- This improves some.
- More improvements can be made



order_items : Table

order_id	item_id	item_qty
125	563	4
125	851	32
125	652	5
126	563	500
126	652	750

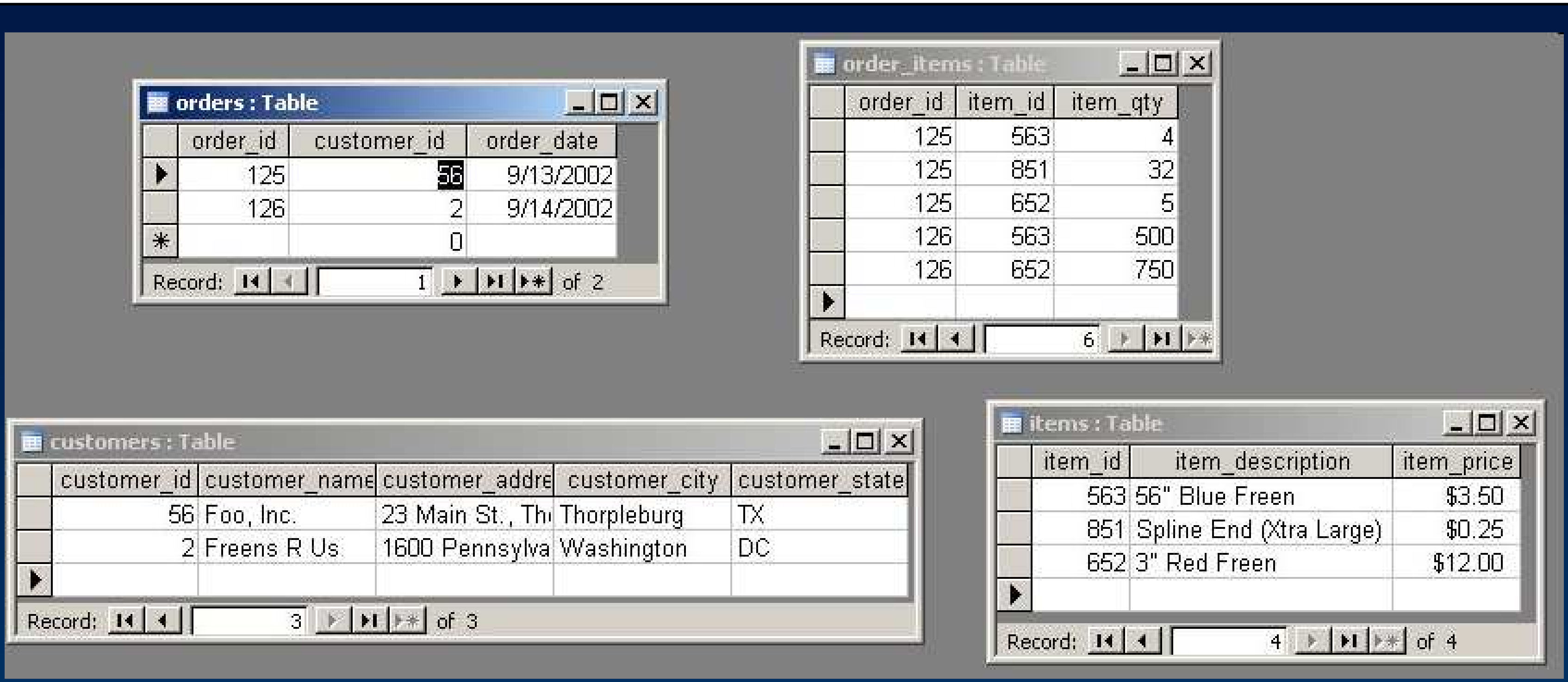
Record: 6



items : Table

item_id	item_description	item_price
563	56" Blue Freen	$3.50
851	Spline End (Xtra Large)	$0.25
652	3" Red Freen	$12.00

Record: 4 of 4



orders : Table

order_id	customer_id	order_date
125	56	9/13/2002
126	2	9/14/2002
	0	

Record: 1 of 2

order_items : Table

order_id	item_id	item_qty
125	563	4
125	851	32
125	652	5
126	563	500
126	652	750

Record: 6

customers : Table

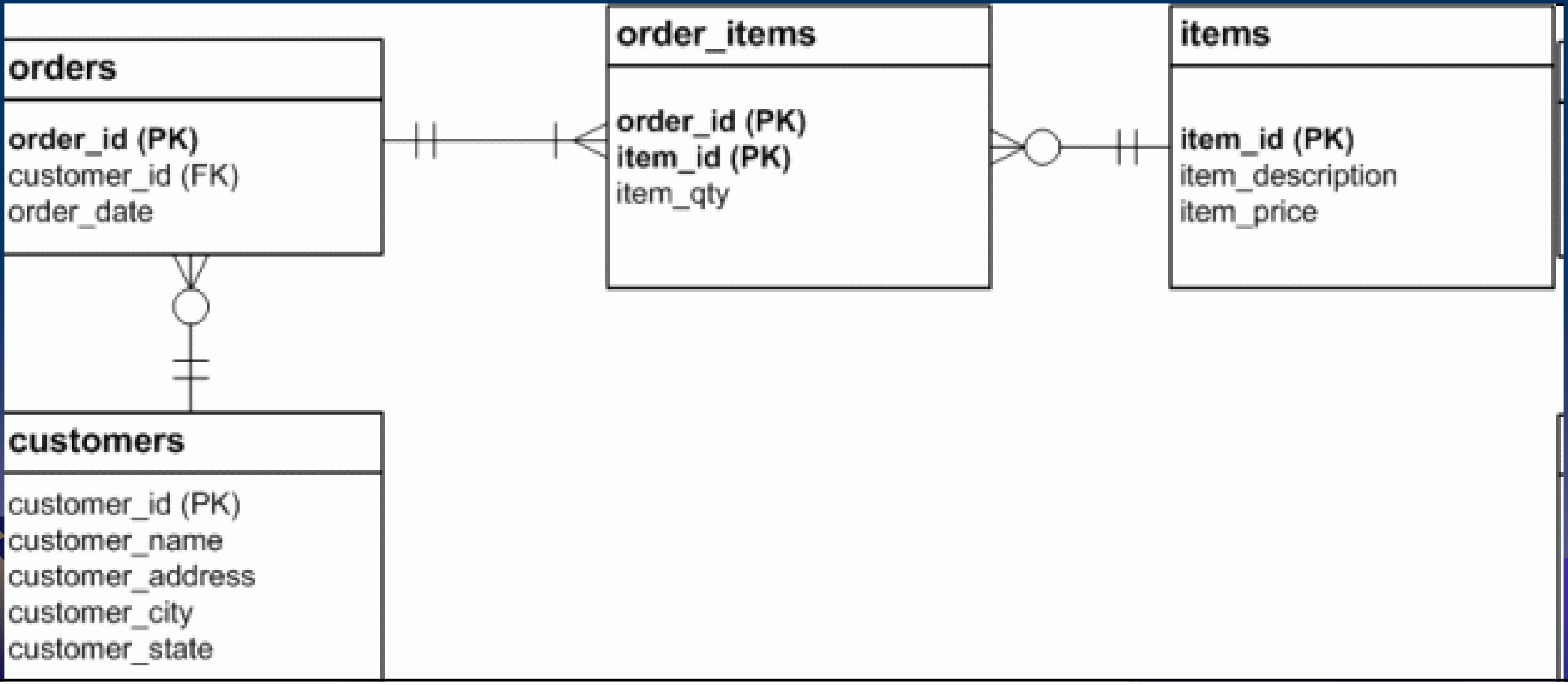
customer_id	customer_name	customer_addre	customer_city	customer_state
56	Foo, Inc.	23 Main St., Th	Thorpleburg	TX
2	Freens R Us	1600 Pennsylva	Washington	DC

Record: 3 of 3

items : Table

item_id	item_description	item_price
563	56" Blue Freen	$3.50
851	Spline End (Xtra Large)	$0.25
652	3" Red Freen	$12.00

Record: 4 of 4



orders

order_id (PK)
customer_id (FK)
order_date

order_items

order_id (PK)
item_id (PK)
item_qty

items

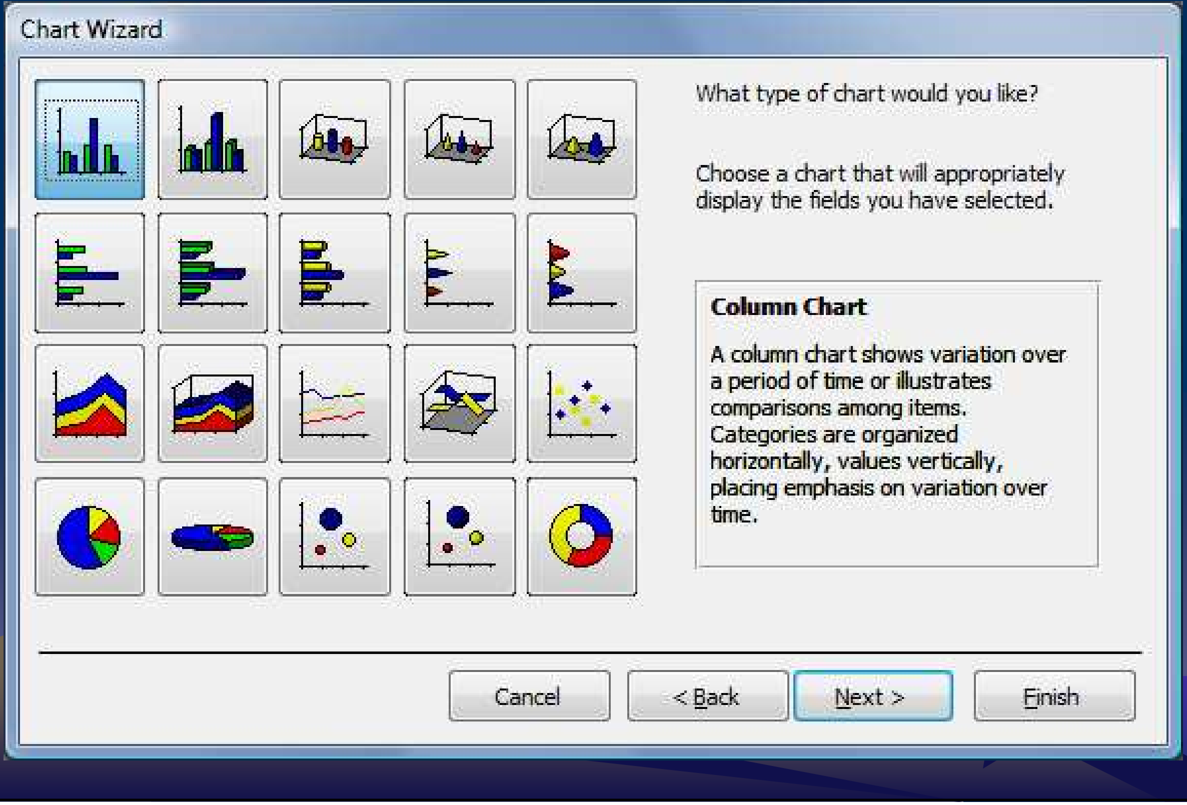
item_id (PK)
item_description
item_price

customers

customer_id (PK)
customer_name
customer_address
customer_city
customer_state

GUI

- Ensuring database normalization will make it easy for programs like Microsoft Access to access the data.
  - Access is a relatively simple interface that will enable engineers to create custom charts and graphs that will suit their needs



What type of chart would you like?

Choose a chart that will appropriately display the fields you have selected.

Column Chart

A column chart shows variation over a period of time or illustrates comparisons among items. Categories are organized horizontally, values vertically, placing emphasis on variation over time.

Risks, Timeline, Conclusion

- Hartman

Risks

- Previous team(2006-2007) was unable to get full system working.
- They were only able to get the interface between the daugterboard and the fpga operational
- We will need to test and debug their interface for full functionality

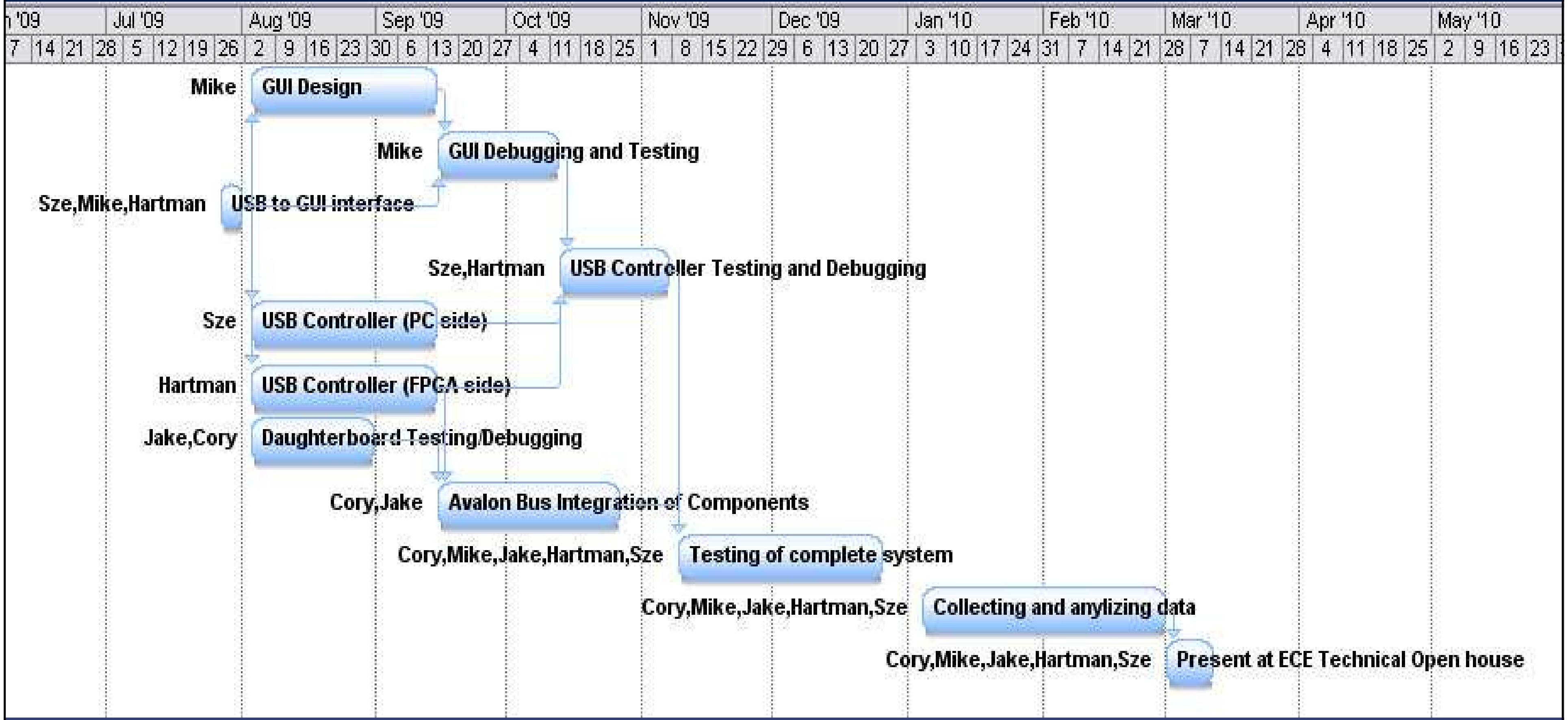
Risks

- The previous team was unable to get USB working properly
- We may still have significant issues with usb connections
  - Hopefully using libusbdotnet will solve this

Risks

- We will need to complete quickly enough so that we can run tests on memory and determine failure patterns and rates

Implementation Timeline



Mike	GUI Design
Mike	GUI Debugging and Testing
Sze,Mike,Hartman	USB to GUI interface
Sze,Hartman	USB Controller Testing and Debugging
Sze	USB Controller (PC side)
Hartman	USB Controller (FPGA side)
Jake,Cory	Daughterboard Testing/Debugging
Cory,Jake	Avalon Bus Integration of Components
Cory,Mike,Jake,Hartman,Sze	Testing of complete system
Cory,Mike,Jake,Hartman,Sze	Collecting and anylizing data
Cory,Mike,Jake,Hartman,Sze	Present at ECE Technical Open house

Conclusion

- Using data generated by the FLAP it will be possible to:
  - Find the best algorithms to correct errors
  - How many spare blocks per chip are necessary
  - Predict failure rates for specific use patterns (server vs. workstation use, etc)