Literate Programming Today

What I did last year

December

Introduction to Artificial Intelligence

Dennis Ruffer

Summary of Online Class

http://www.forth.org/svfig/kk/12-2011-Ruffer.pdf

January and February

Literate Programming
Testable Documentation

My ongoing saga of using Noweb in LyX

http://www.forth.org/svfig/kk/02-2012-Ruffer.zip

March, April, May and June

Initial PWM Test

Dennis Ruffer

18 May 2012

Abstract

These are some notes from my 1st experiences on Green Array's Evaluation Board (EVB001 rev 0.1.1). I will try to explain things precisely, so that others can follow after me. Although I have worked with earlier versions of Chuck's chip, I have not used the ColorForth IDE. This means that I am learning just about as much as most readers of this document and, as they recommend in the arrayForth User's Manual (DB004 Revised 12/23/11), I will be making small changes, saving often and testing every change.

http://www.forth.org/svfig/kk/05-2012-Ruffer.zip

July and August

DataBase Support System

Elizabeth Rather and Dennis Ruffer

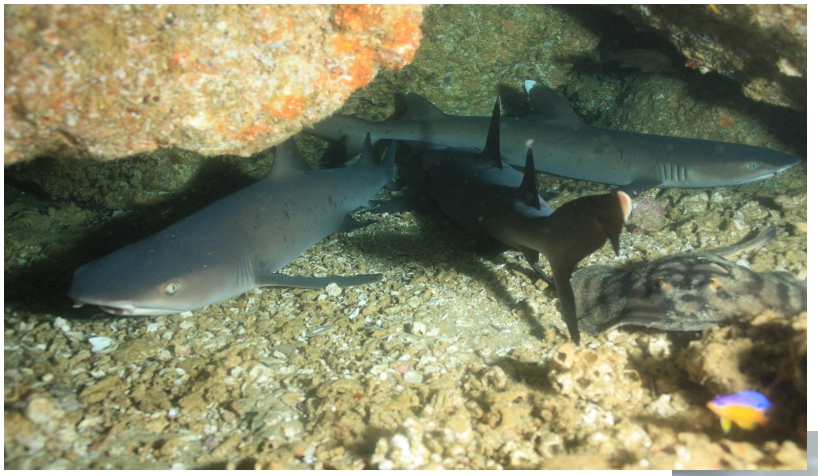
08 July 2012

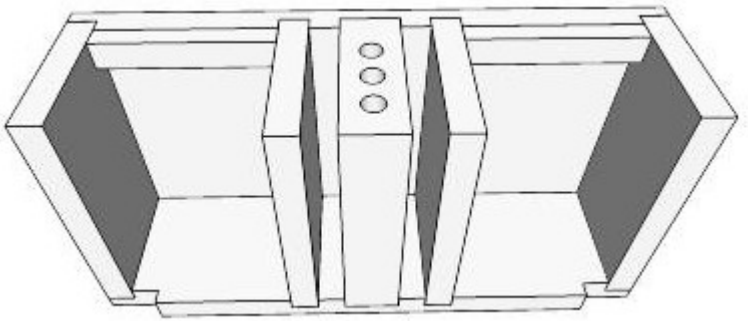
Abstract

To describe a simple database support system. This system originated as a loadable module in polyFORTH. It was converted to be compatible with the ANS Forth standard with a goal to become an IEEE 1275 binding for a database package. Later, it was put into the VentureForth development system for various purposes, including the Timing Diagrams. Now, it is being used again in the colorForth assembler/disassembler and converted to be compatible with 64-bit gforth, using Literate Programing in LYX

http://code.google.com/p/vf-plugins/downloads/detail?name=pFDatabase-5.zip

September and October





ColorForth Assembler/Disassembler

I took over this program originally from John Rible as an initial contract with IntellaSys in Oct. 2006, with the purpose to convert colorForth (cf) source into ASCII text and then added Albert van der Horst's ciasdis to illuminate the cf kernel. The goal is to produce a complete round trip, reproducing the cf file with an ASCII disassembly (dsm) file that can be used in its own Literate Programming document.

Color Blindness

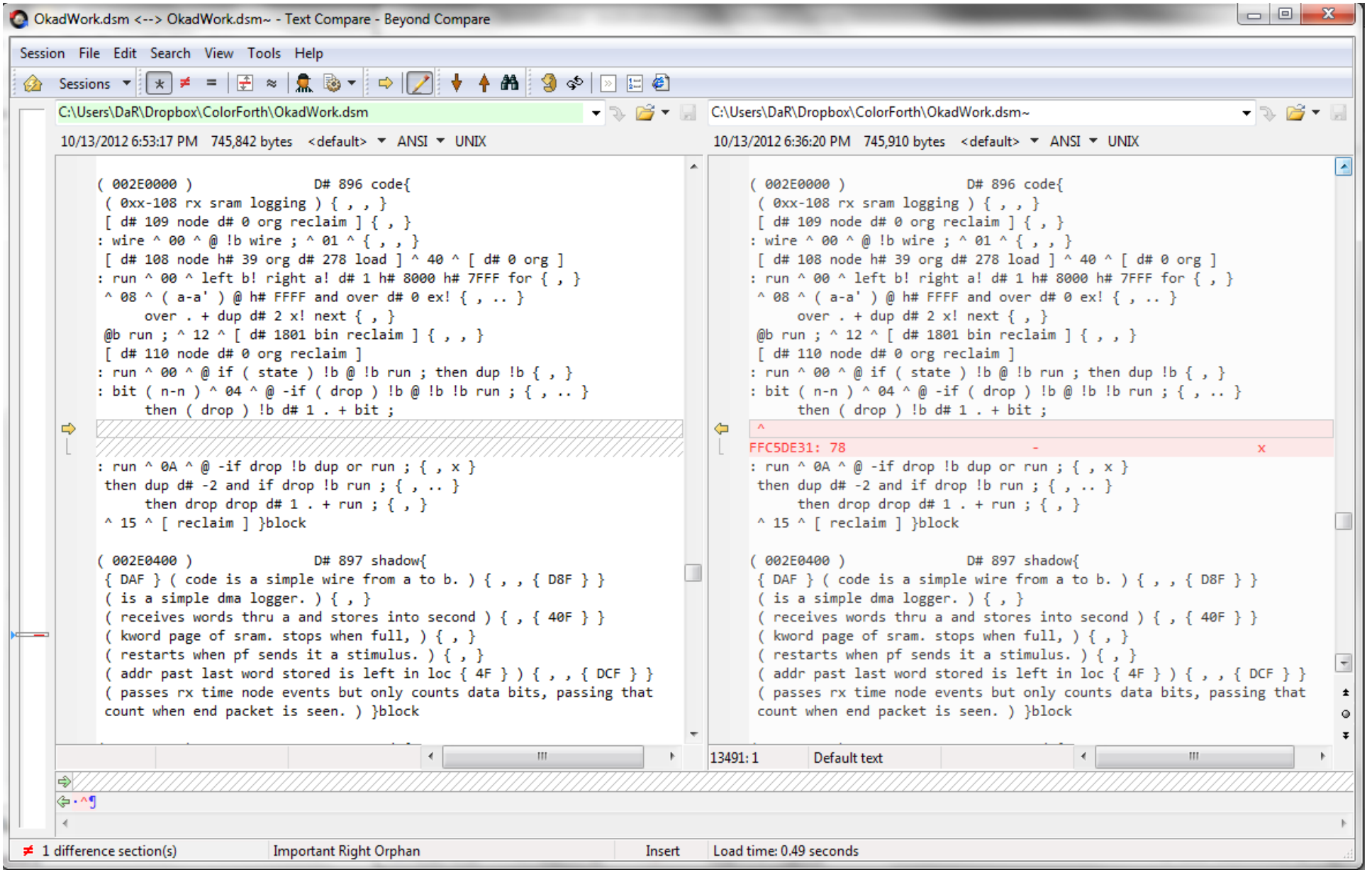
Before I start diving much deeper, I should explain that I am red/green color blind. This means that I do not see, or react to color like most people do. 7% of males have this condition, as well as other people who do not perceive color the same way as others do. I have used a program called **eyePilot** (Version 1.0.12 from Tenebraex) so I can figure out what colors ColorForth is using, but it has been withdrawn from the market. So, most recently, I found **WhatColor** http://www.hikarun.com/e/, which appears to be supported, at least. I don't always need it, but frequently, yellow and green look far to much alike and on the block I will be using in a few moments, I see that the green component can have an RGB value of 192 or 255. I see that the User's manual explains that these are HEX numbers, but this does not make using colorForth any easier for me.

colorForth → ASCII

- [r] (Red text)
- [w] (White text)
- [g] (Green text)
- [y] (Yellow text)
- [m] (Magenta variable)
- [b] (Blue text)
- [c] (Cyan text)
- [a] (Address in grey)

colorForth	ASCII translation
red	: red
white	(white)
green	: ... green ... ;
yellow	[yellow]
magenta	:# magenta
blue	{ blue }
cyan	< cyan >
grey	^ grey ^

Today


OkadWork.dsm <--> OkadWork.dsm~ - Text Compare - Beyond Compare
Session File Edit Search View Tools Help
Sessions
C:\Users\DaR\Dropbox\ColorForth\OkadWork.dsm
10/13/2012 6:53:17 PM 745,842 bytes <default> ANSI UNIX
C:\Users\DaR\Dropbox\ColorForth\OkadWork.dsm~
10/13/2012 6:36:20 PM 745,910 bytes <default> ANSI UNIX
(002E0000) D# 896 code{
(0xx-108 rx sram logging) { , , }
[d# 109 node d# 0 org reclaim] { , }
: wire ^ 00 ^ @ !b wire ; ^ 01 ^ { , , }
[d# 108 node h# 39 org d# 278 load] ^ 40 ^ [d# 0 org]
: run ^ 00 ^ left b! right a! d# 1 h# 8000 h# 7FFF for { , }
^ 08 ^ (a-a') @ h# FFFF and over d# 0 ex! { , .. }
over . + dup d# 2 x! next { , }
@b run ; ^ 12 ^ [d# 1801 bin reclaim] { , , }
[d# 110 node d# 0 org reclaim]
: run ^ 00 ^ @ if (state) !b @ !b run ; then dup !b { , }
: bit (n-n) ^ 04 ^ @ -if (drop) !b @ !b !b run ; { , .. }
then (drop) !b d# 1 . + bit ;
: run ^ 0A ^ @ -if drop !b dup or run ; { , x }
then dup d# -2 and if drop !b run ; { , .. }
then drop drop d# 1 . + run ; { , }
^ 15 ^ [reclaim] }block
(002E0400) D# 897 shadow{
{ DAF } (code is a simple wire from a to b.) { , , { D8F } }
(is a simple dma logger.) { , }
(receives words thru a and stores into second) { , { 40F } }
(kword page of sram. stops when full,) { , }
(restarts when pf sends it a stimulus.) { , }
(addr past last word stored is left in loc { 4F }) { , , { DCF } }
(passes rx time node events but only counts data bits, passing that
count when end packet is seen.) }block
FFC5DE31: 78 - x
13491: 1 Default text
1 difference section(s)
Important Right Orphan
Insert
Load time: 0.49 seconds

Load LyX file directly into gforth

```
: $CREATE ( a n -- )   S" CREATE " DUP >R  'CREATE SWAP MOVE
   'CREATE R@ + OVER >R  2DUP + CREATE' > ABORT" Name too long!"
   SWAP MOVE  'CREATE R> R> + EVALUATE
   DOES>  DUP 2 CELLS + 2@  ROT 2@
      2OVER 2OVER D< ABORT" Unfinished symbol"
      push-buffer  nw-file-id @ REPOSITION-FILE THROW  BEGIN
         file-input-buffer 1024 nw-file-id @ READ-LINE THROW
         0= ABORT" Truncated symbol"
         CR file-input-buffer OVER TYPE
         file-input-buffer SWAP EVALUATE
         2DUP nw-file-id @ FILE-POSITION THROW D<=
      UNTIL  R> DROP 2DROP  pop-buffer ;
```