Appendix A: *Hoxb1* perturbation

Perturbation experiments have been well worked out in some systems (most notably yeast and *Drosophila*), but they are harder to do in higher organisms. This is not to say that they are impossible, and the following section describes an experiment that was designed to investigate a component of the *Hox* network.

As mentioned previously, a group has presented a model that asserts that *Hoxa1* and *Hoxb1* are involved with the repression of *Krox20* (Barrow et al., 2000). This supposition was implemented in the baseline model presented in Chapter 3. However, there are reasons to believe that their model might not be accurate. In the $Hoxb1^{null}$ mutant there are no changes in the level of *Krox20*, and in the $Hoxa1^{null}/Hoxb1^{null}$ double mutant embryos there is no sign of *Krox20* in rhombomere 3 and reduced expression rhombomere 5. In addition, the $Hoxa1^{null}$ mutant mouse shows reduced levels of Krox20 in rhombomere 5 (Gavalas et al., 1998; Studer et al., 1998). Part of the difficultly in interpreting these results is that the rhombomeres in these mutants are often altered. For instance, in the $Hoxa1^{null}/Hoxb13'RARE^{null}$ mutant, a territory with new characteristics forms in the place of rhombomere 4 (Gavalas et al., 2001). But the rhombomere alteration does not always occur; rhombomere 3 appears to be normal (using both visual and *in situ* assays) in the $Hoxa1^{null}/Hoxb1^{null}$ double mutant (Studer et al., 1998). Taken together, the evidence does not seem to support the model of *Hoxb1* and *Hoxa1* playing a role in repressing *Krox20*. In an effort to test this conjecture, an experiment was designed to perturb the system using a specially designed piece of DNA. This

experiment would directly address the predicted changes in *Krox20* expression due to the knockout of *Hoxb1*

A detailed study of the *Drosophila* protein Engrailed showed that it was able to repress transcription activity (Han and Manley, 1993), and it has been fused to other proteins to provide a dominant negative like activity in a system. However, this fusion has been done primarily in *Xenopus* and fish (*cf.* LaBonne and Bronner-Fraser, 2000; Vignali et al., 2000).

Starting with the CS2+ vector (R. Rupp and D. Turner), Heather Marshall from the Stowers Institute for Medical Research inserted the cDNA for *Hoxb1* into the polylinker between the BamH1 and Xho1 sites. Into this construct she inserted the Engrailed repressor in frame at the unique XmaIII site of *Hoxb1*. The modified protein with the Engrailed repressor would attach to the *Hoxb1* binding domain and would in turn repress the expression of *Hoxb1* due to the auto regulatory loop. In addition, it would repress any gene that *Hoxb1* could attach to.

Her original plan for this construct was to use the construct for mRNA fish injections. This is a relatively easy procedure for a variety of reasons, not the least of which is that one cell fish embryos are easily harvest and manipulated. But, after making this construct, Dr. Marshal did not ever use it in fish and she provided it to the author for use in a chick perturbation experiment.

Introducing this DNA in a way that it becomes active would be a fantastic test of the model. If the DNA could in fact repress *Hoxb1* expression, then assaying for *Krox20* would allow for another piece of evidence concerning the supposed *Hoxb1-Krox20* connection.

On the modeling side, the results of the *Hoxb1*/Eng construct produces results that are very similar to those of the *Hoxb1* mutant in Chapter 3. This is evident in the results shown in Fig A.1 below.

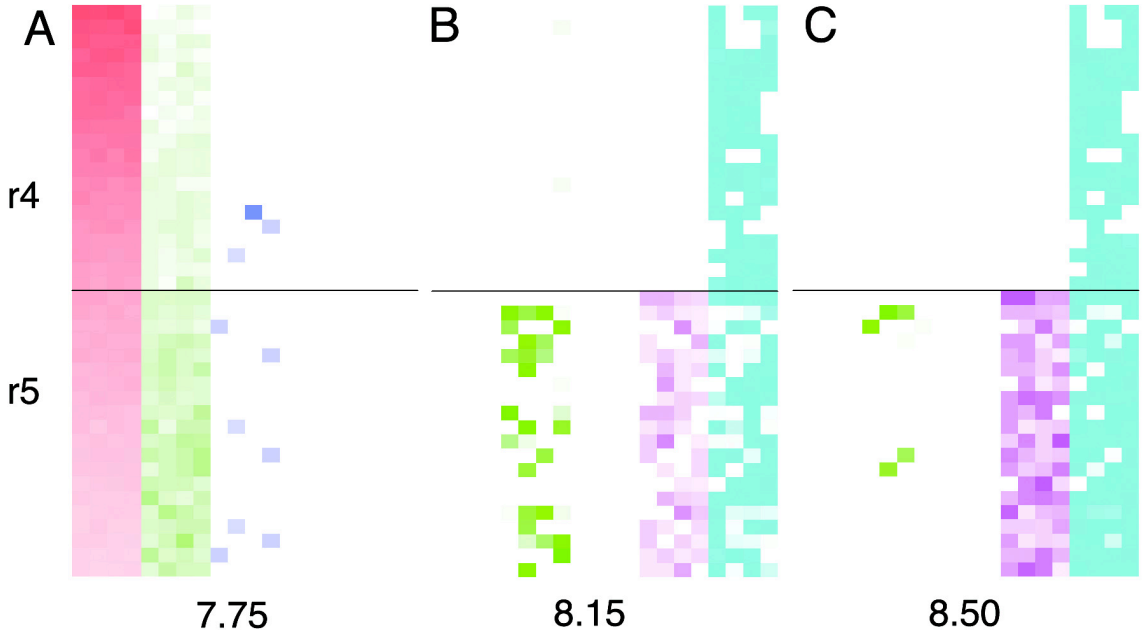

Figure A.1 *Hoxb1*/Eng model results. These results are very similar to the *Hoxb1* mutant presented in Chapter 3 (Figure 3.5). There is a near total down-regulation of *Hoxb1* which, combined with the normal fading of *Hoxa1*, allows for the up-regulation of *Krox20* in rhombomere 4. *Hoxb2* does appear in r5 due to the *Krox20* up-regulation, but is absent in r4 because of the lack of *Hoxb1*. These simulations were run before making the change to the mechanism for transcribing *Hoxa1* brought about by the work described in Chapter 4.

The similarities between the *Hoxb1* mutant in Chapter 3 and the *Hoxb1*/Eng construct are not unexpected at all. In both cases the dominant effect is the repression of

Hoxb1. However, in the *Hoxb1*/Eng construct, there is an occasional low level of the *Hoxb1* product still as not all of the system would be bound. Therefore, the lack of *Hoxb1* and *Hoxa1* would result in an expansion of *Krox20*. It is this expansion of *Krox20* that the experiment was designed to test.

Electroporation

Introducing the *Hoxb1*/Eng construct cannot be accomplished in the same manner as a 1-cell fish injection, but a different technique that accomplishes the same result, namely having the foreign DNA incorporated into the organism, can be undertaken. Electroporation is a technique for introducing foreign DNA into cells. The method involves breaking down the membrane of cell walls through the use of an electric pulse. In addition to creating holes in the membrane, the electrical gradient drives the negatively charged DNA into the holes in the membrane. There are a variety of variables that contribute to the effectiveness of the electroporation including the size and placement of the electrodes, but by far the most important component is the duration and voltage of the pulse. In general, 3-5 pulses of 50 microsecond duration and between 7 and 25 volts works well. Excellent technical reviews can be found in (Itasaki et al., 1999; Swartz et al., 2001). One very nice benefit of the electroporation is that because of the electrical gradient, only cells on the positive side of the neural tube have their cell walls broken down, and only one half of the embryo is exposed to the DNA. This provides an excellent internal control, as one side of the embryo is experiment, and the other side is normal.

Electroporation can be a tricky procedure, and in order to check that the electroporation worked correctly, an additional control was required. To this end, an IRES GFP construct was purchased from Clontech. The internal ribosome entry site (IRES) is a sequence of DNA that a ribosome recognizes and will attach to. This leads to the creation of green fluorescent protein (GFP) mRNA, and when it is translated into proteins, a cell that has incorporated this DNA will glow when excited with the proper wavelength of light.

To create this construct, the Clontech IRES-GFP module was removed using the restriction enzymes XhoI and XbaI then cloned into the *Hoxb1* Engrailed construct just after the stop codon for the *Hoxb1*. The result is a bi-cistronic message: one that has two gene products (in this case the *Hoxb1*/Eng repressor and the GFP) from adjacent stretches of the same mRNA. In general the message from the second coding region will be weaker than the first. If the electroporation is successful, the GFP will be glowing and that also signifies that the *Hoxb1*/Eng fusion protein has been created. It is important to remember that it does not provide any information about if the *Hoxb1*/Eng repressor message is working correctly.

Embryos

Fertile pathogen free chicken eggs were acquired from Charles River Laboratories and incubated at 38°C until the proper stage of development, usually between 28-34 hours. The eggs were rinsed with 75% alcohol and 3 mL of albumin was removed. The eggs were windowed and a solution of .1% food coloring (equal amounts of FD&C Red

40 and FD&C Blue 1 from Spectra Colors Corp.) in Hanks' Balanced Salt Solution (HBSS) was injected beneath the blastoderm to provide contrast.

After windowing the eggshell and injecting the dye, a small amount of HBSS was added to the top of the embryos to keep it moist. A tungsten needle was then used to cut a hole in the vitelline membrane near the hindbrain of the embryo. A solution of ~1ug/ul of the Hoxb1/Eng/IRES GFP construct (with a small amount of Fast Green dye included to make the injection easier to see) was injected using a quartz micropipette into the lumen of the neural tube. Electrodes made from platinum wire were laid flat on the area opaca, parallel and lateral to the embryo. About 1-2 mm of contact was made with the area opaca, and the electrodes were approximately 5 mm apart. 3-5 current pulses of 20-40 V and 50-100 ms duration were applied. More HBSS or Ringer was added to the top of the embryo, the egg was resealed with packing take and placed back into the incubator for 8 hours. The embryos were then screened for fluorescence using a Leica microscope. Positive embryos (Figure A.2) were harvested and fixed in 4% paraformaldehyde (PFA) solution overnight at 4°C. The next day they were dehydrated through a series of methanol washes, and were placed in a –20°C freezer for storage. Embryos stored in this manner can be kept in a freezer for upwards of a year, but in this case they were not in the freezer for more than a couple months. After storage, the embryos were re-hydrated into phosphate buffer saline (PBS) and whole mount *in situ* hybridization was performed.

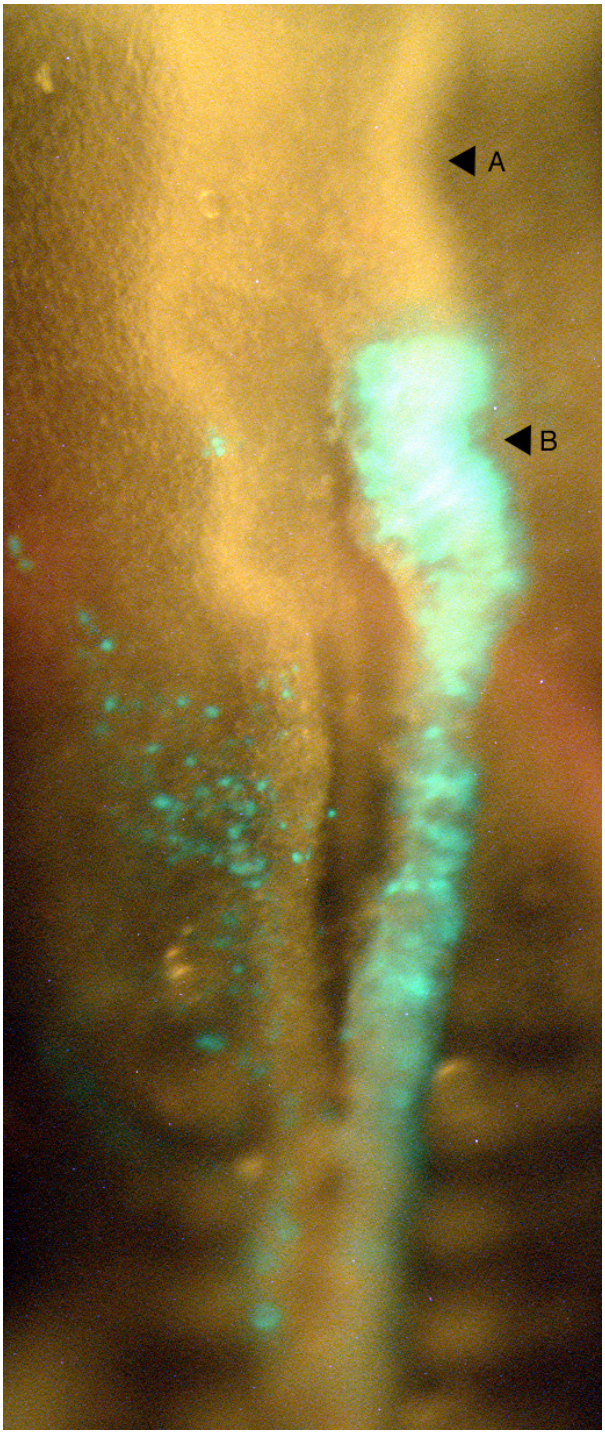

Figure A.2 Glowing hindbrain. This picture shows a 6.3x magnification of the hindbrain of a stage 11 embryo after electroporation at stage 7. This picture was taken *in ovo* during the screening process for embryos in which the *Hoxb1*/Eng/GFP construct was successfully incorporated into the cells during

electroporation. The midbrain/hindbrain boundary is marked by **(A)**, and the boundary between the third and fourth rhombomeres is marked by **(B)**. Despite the name of GFP, the cells in this instance were colored cyan in Photoshop to provide contrast to the anatomy of the embryo.

This experiment was performed using the embryos that fluoresced strongly (as in Figure A.2) and after performing the *in situ* for *Hoxb1*, the gene that would be affected if the construct were working correctly, there was no visible difference between the control and the experimental embryos.

With the negative results from this experiment, it became clear that something wasn't working right, but it could be a variety of things. First of all, the construct may not be working at all, or it may not have been introduced into the embryo at an early enough time point. If this occurred, the *Hoxb1*/Eng construct would not be able to shut down the system that was already adequately initiated. The later problem was the logical one to test, and it was initially presumed to be easier. It turns out that it was anything but.

In an experiment initially performed by Kristen Correia at the Stowers Institute, the original construct with the CMV enhancer was electroporated into chicks at HH stage 4 (Hamburger and Hamilton, 1951). This experiment is very different than the one described earlier as there is no neural tube to hold the DNA solution. The entire procedure is described below.

After performing a dozen of these electroporations, it was reported that there was only very low level of fluorescence. Further investigation into this led to the conclusion that the CMV enhancer doesn't work very well at these early stages of development, and

the CS enhancer should be used instead. CS is a combination of the CMV enhancer and the Chick Beta actin promoter.

The backbone of his LZRS-CA-H2B-YFP construct (originally used for the creation of a retrovirus for infection of quail) was the basis for the new Hoxb1/Eng repressor (courtesy R. Lansford). The cloning was done in two stages. The H2B-YFP module was removed using the NotI and XhoI enzymes, and the Clontech IRES GFP module was inserted into the LZRS-CA backbone after the band was isolated using a double digestion with XhoI and Not1. At this point the Hoxb1/Eng module was removed from the CS2+ construct using BamH1 and XhoI, commercially available Xho linkers (NE Biolabs) were added, and the resulting DNA fragment was cloned into the Xho site of the LZRS-CA-IRES-GFP. In yet another example of the difficulty in laboratory work, the work described in this paragraph took over six weeks to perform.

The construct (hereafter called CS-*Hoxb1*/Eng) was tested by transfection into two different cell lines and the results are shown in Figure A.3.

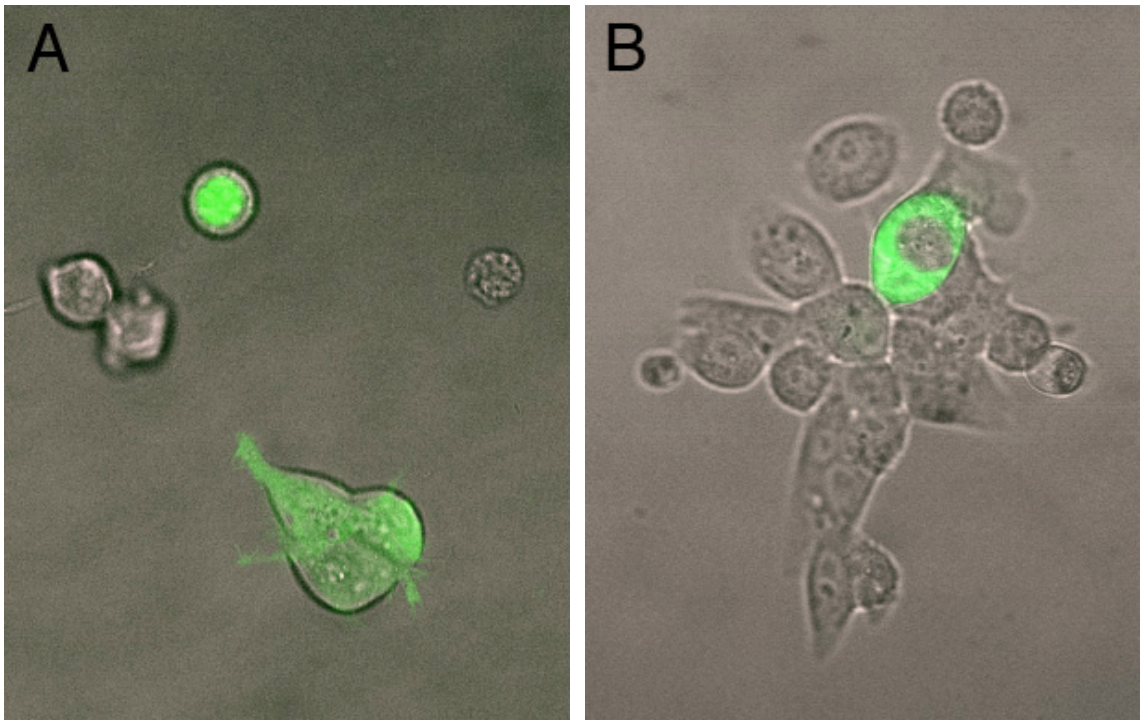

Figure A.3 Transfected cells. 63x magnification pictures of **(A)** Chinese hamster ovary and **(B)** 293 GPG cells transfected with the CA-*Hoxb1*/Eng construct. The cells were transfected using a 1 μ gram/μ liter DNA solution combined with Superfect (Qiagen). Superfect is a specially designed dendrimer that forms a complex with the DNA of interest and enters the cell using negatively charged receptors (Qiagen, 2000; Tang et al., 1996). Both these images were acquired on a Zeiss 410 inverted microscope.

The successful glow to the cells was promising, and the author visited the Stowers Institute for medical research to learn how to electroporate chick embryos at a young age. As mentioned previously, the procedure for this type of electroporation is much more difficult since there is no neural tube to contain the DNA. Instead, the DNA must be injected between the vitelline membrane and the embryo directly over the node. If the

DNA spreads beyond the translucent border of the embryo, it was injected on top of the vitelline membrane. If the DNA is localized in a small area, the yolk was injected.

The electroporation must be done with electrodes oriented such that the positive terminal is inserted into the yolk beneath the embryo, and the negative terminal is on top of the grove containing the DNA. Figure A.4A shows the major landmarks in a stage 4 embryo which provides an orientation for this process, and Figure A.4 B is a picture of the custom electrodes that were created for the electroporation.

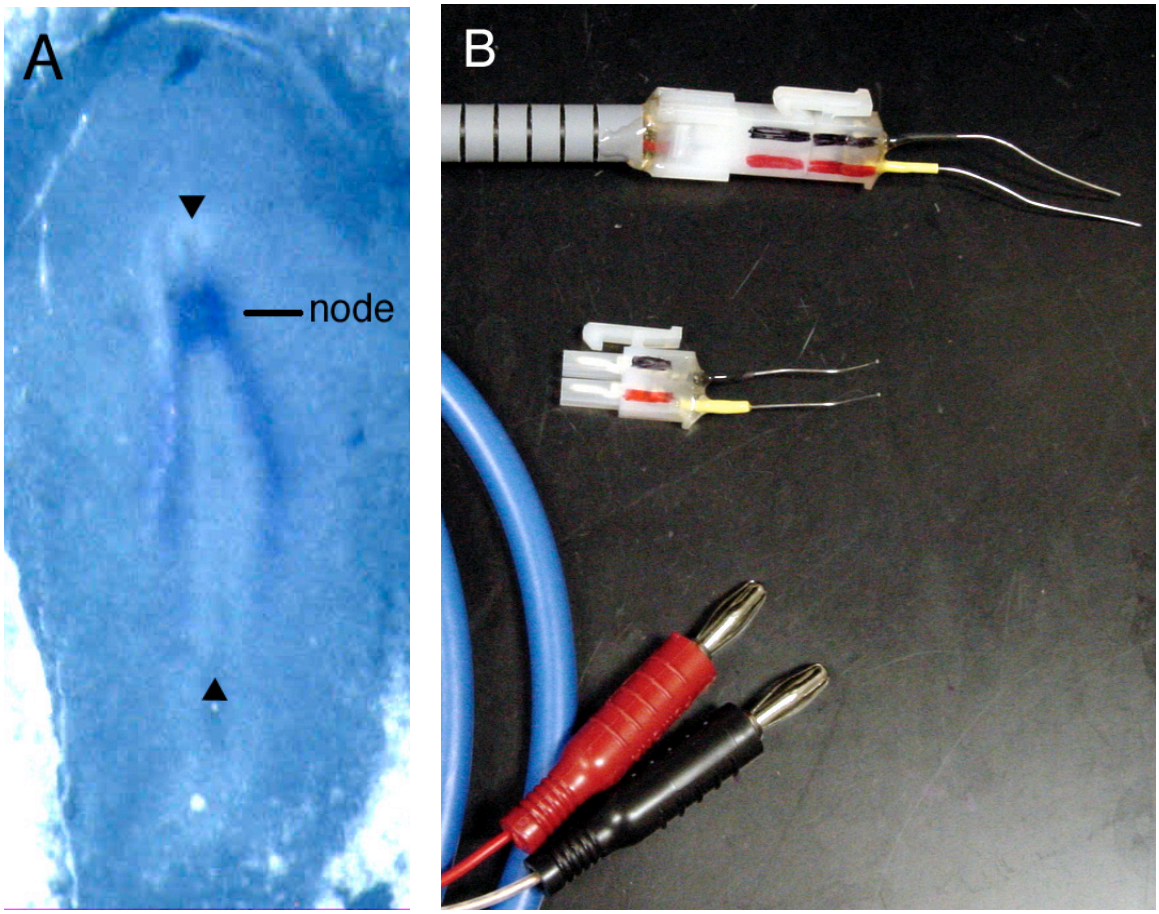

Figure A.4 Stage 4+ embryo. (A) This embryo, stained for the gene *Fringe* (probe courtesy of C. Tabin), highlights the important landmarks of a late stage 4 embryo. Anterior is towards the top of the page, posterior is toward the bottom. The primitive groove extends along the anterior/posterior axis between the arrows. Hensen's node (which is strongly expressing *Fringe*, as evidenced by the dark blue color) is clearly marked, and the neural folds (which eventually fold over and join together to become the neural tube) are on either side of the neural grove and are also expressing Fringe toward the anterior part of the embryo **(B)** Electrodes that were used for the electroporation experiments. These were built using 24 gauge platinum wire from a Caltech supply room, and others parts from a local electronic supply (MarVac). The electrodes are designed in such a way to make the configuration easily changeable for the proper application. The electrodes at the top are for stage 4/5 electroporation in which the electrodes need to be above and below the embryo, while the electrodes in the middle of the picture are for later stages in which the neural tube is more fully formed.

A construct containing CA driving GFP was used as a control for testing the efficacy of the technique using the equipment in the lab. An example of a control embryo is shown in Figure A.5.

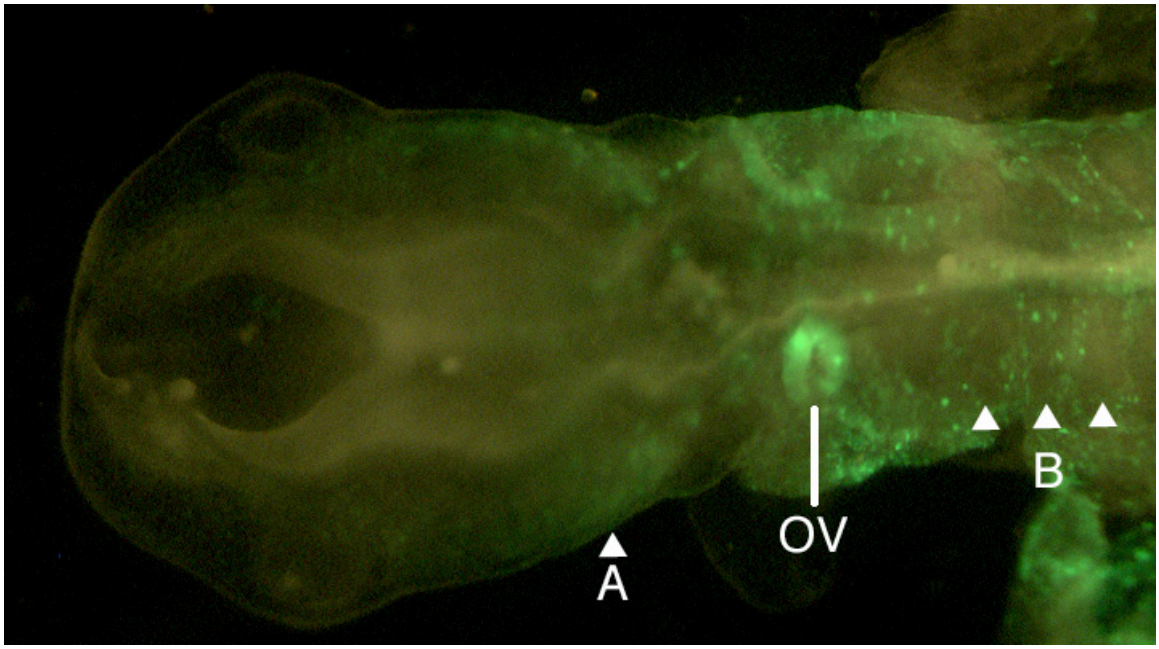

Figure A.5 CA-GFP embryo. This embryo was electroporated with CA driving GFP at stage 4 and collected at stage 12+. The early electroporation was done in the primitive streak and the node and the neural cells are strongly labeled. This picture is a 6.3x view of taken under a long pass GFP filter that allows both the fluorescent and white light to pass. The otic vesicle (OV) is clearly highlighted, as are streams of neural crest cells populating the head (A). A collection of vertical cells at B rings three somites. Note that the expression is much stronger than the hindbrain in Figure A.2.

Over 70 embryos were electroporated with the CS-*Hoxb1*/Eng construct at the Stowers Institute, but none of them glowed under the fluorescent scope. This was problematic, especially considering the results of the tissue culture tests, but there are several possible reasons why they didn't glow. The most likely one is that the second

message from the bi-cistronic structure wasn't effective. Despite the lack of fluorescence, the embryos were harvested and prepared for *in situ* hybridization as described in Chapter 4.

The probe used was *Hoxb1* (courtesy of Robb Krumlauf), and the initial results were potentially promising as they looked very different from wild type, both seen in Figure A.6.

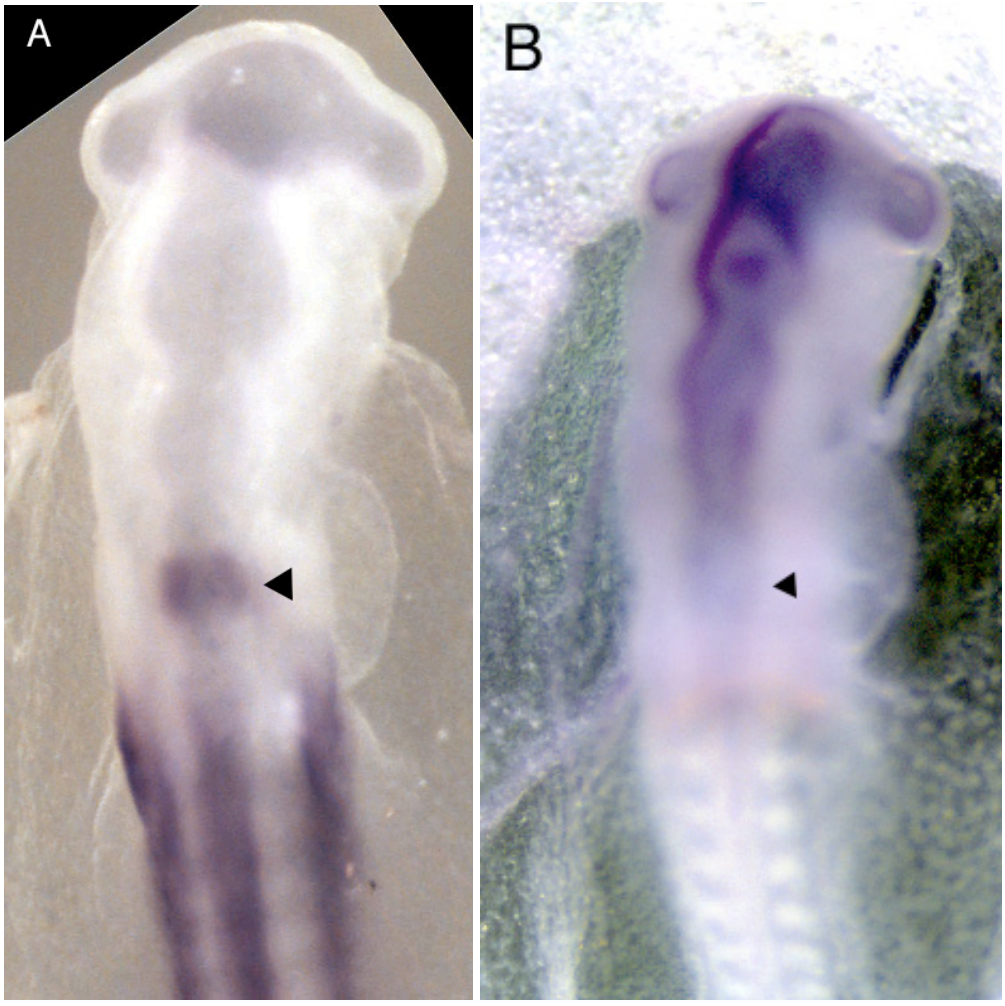

Figure A.6 *Hoxb1* expression in wild-type (A) and **CS-*Hoxb1*/Eng (B)** These embryos, stained for the gene *Hoxb1* shows a potential difference in the control and experimental embryos. In the wild-type embryo, the r4 (marked by the black

arrow) and neural tube expression is clearly seen. In addition, there is no expression in the mid or forebrain which is as expected. In the experimental embryo in **(B)** there is no expression in r4 (marked by the black arrow), no expression in the neural tube, but there appears to be expression in the mid and forebrain.

This pattern seen in Figure A.6 was visible in a half dozen embryos, but there were some concerns that the pattern was not real and was, in fact, an artifact brought about by a staining condensate that collected in the neural tube. To check this concern, an antibody treatment was performed on the embryos. The antibodies used would probe for GFP and *Krox20*. The GFP antibody (courtesy of H. McBride) was used to test if the electroporation of the construct worked, as it might be the case that the level of expression of the GFP was just too low to visualize with light microscopy. The *Krox20* antibody (purchased from Covance) was used as the experimental assay, as it would test the connection between the *Hoxb1* and *Krox20*.

For the antibody procedure, the embryos were rinsed in PBS 3 times for 5 minutes each. The PBS was then replaced with a 1:50 concentration of primary antibody in PBS and rocked at room temperature for 2 hours. Another set of 3 PBS rinses followed, followed by a 1:50 concentration of secondary antibody in PBS. After two hours of room temperature rocking with the secondary, the embryos are given their final set of PBS rinses. The secondary antibody contained the fluorescent tag CY-3, so the embryos were screened under a fluorescent microscope.

The GFP antibody worked well against the control embryos that had been electroporated with CA-GFP, and there was clear co-localization between the different colors. However, there was no sign of fluorescence from the experimental embryos that were electroporated with CA-*Hoxb1*/Eng. In addition, the *Krox20* antibody didn't work in any of the embryos, control or experiment. This could be for a variety of reasons, but the first line of investigation should be the fixation process. For instance, when using HNK-1, an antibody which stains migratory neural crest cells, it has been determined that anything more than a 10-minute fix in 4% PFA solution will cause the staining to fail, as will using Bouins' fixation (H. McBride, personal communication). This is unlike *in situ* hybridization in which the amount of fixation time or the fixative used is not an issue. Indeed, leaving embryos in 4% PFA for 2 days at room temperature does not noticeably affect the *in situ* hybridization (data not shown).

While considering the next step to take in this investigation, caveats to this experiment appeared from Andy Groves, a former post-doc at Caltech who is now at the House Ear Institute. Using a construct that is very similar to the CS-*Hoxb1*/Eng repressor, Dr. Groves used a Dlx engrailed repressor construct to look at ear formation. After electroporating this construct in at stage 4, they observed a very clear phenotype: the failure to form a proper ear. As a control they created a variation of their Dlx-engrailed-repressor construct that cannot bind DNA due to point mutations in their homeobox. It was a surprise and a disappointment to discover that the same phenotype appeared. Dr. Groves is still investigating this phenomenon, and has been provided the CS-*Hoxb1*/Eng construct to use in his experiments.

This means that a rigorous control for the CS-*Hoxb1*/Eng experiment would require the electroporation of a construct with a point mutation in the homeobox. This fact, combined with the numerous issues that still required troubleshooting, forced this experiment to be put on hold. It should certainly not be considered a failure, as significant steps have been accomplished in making this new construct to test this connection between *Krox20* and *Hoxb1*.

Appendix B: Protocols

Whole Mount In Situ *Hybridization*

This protocol is designed for young (<20 somites) chick embryos that do not need to be treated with Proteinase K. It is based on a protocol supplied by Helen McBride who also drew upon information from Reinhard Koester, Andy Groves, and David Wilkinson (Wilkinson, 1992). All solution volumes (except the pre-hyb solution) are the amount needed for each vial being processed, assuming that no more than 10 ml will be used for each wash. Multiply accordingly if necessary.

General Comments

For performing *in situs*, there are a few ways to handle the samples. Doing everything with 15 ml Falcon tubes is possible, but defiantly the archaic way to go. 12 well culture chambers are nice in that it is much easier to transfer liquids in and out. In addition, empty wells can be used to discard your waste liquids so you can rescue any embryos that may have inadvertently been sucked up. Another way is to use Reactivials

with the cover insert replaced by a fine nylon mesh (Shandon biopsy bags), a method developed by the author. This fluid can then be poured out without losing the embryos. In addition, a suction device can be used, but be warned, it is still possible to suck up the embryos through the mesh. Forcing liquid back in with a pipet is easy. The embryos can stick to the bag, but if this happens, they can be pushed back into the tube with fluid pressure.

Day 1 Rehydration and Hybridization

Solutions

PBT

Tween 20	.1%	500ul
1x PBS		500ml
Final Volume		500.5ml

Use PBT the same week as you add the Tween.

Pre-hybridization solution

Formamide	50%	25 ml
Depc SSC	5x	12.5 ml 20x Depc SSC
yeast RNA	50ug/ml	125 ul of 20 mg/ml tRNA
SDS	1%	.5 g
Heparin	50 ug/ml	.0025 g
Depc H_2O		12.375 ml
Final Volume		50 ml

Pre-hyb mix can be stored for several weeks at -20°C. Make sure to use SSC made with Depc H_2O.

Hybridization solution

Pre-hybridization buffer with probe added. A final volume of 1 ug probe per ml is typical. My probes are washed off a Qiagen column with 50 ul water and added to 150 ul of pre-hyb. I then add 2 ul of this to 100 ul of pre-hyb to make the hybridization solution.

Protocol

1) Re-hydrate embryos through a Methanol series (75%; 50%; 25% Methanol/PBT) for 5-20 minutes each and wash 2x 5 minutes in PBT.
2) Add .5 ml pre-warmed pre-hyb buffer and swirl embryos around.
3) Replace with 1 ml warmed pre-hyb buffer and let rock at 65°C for 1-2 hours.
4) Replace with .5 ml of hyb solution and rock overnight at 65°C.

Day 2 Post-hybridization Washes and Antibody Incubation

Solutions

Wash Solution 1

Formamide	50%	15 ml
SSC, pH 4.5	5x	7.5 ml 20x SSC
SDS	1%	3 ml 10% SDS
ddH20		4.5 ml
Final volume		30 ml

Wash Solution 2

Formamide	50%	15 ml

SSC, pH 4.5	2x	3 ml 20x SSC
SDS	.2%	600 ul 10% SDS
ddH20		11.4 ml
Final volume		30 ml

Mab+Lev; Tween

Maleic Acid disodium salt	100 mM	2.4 g
NaCl	150 mM	1.315 g
Tween 20	.1%	150 ul
Levamisole	2 mM	.07224 g
ddH20		150 ml
Final volume		150 ml

Add the Levamisole and Tween 20 on the day of use and filter. Levamisole is a phosphatase inhibitor that should inhibit the native alkaline phosphatase and thus reduce the background, but opinions vary as to the effectivenes of this treatment. In general, it won't hurt, but it you forget to add it, you may not even notice the difference. The Mab solution can be used the next day.

Antibody (Ab) block solution

Blocking Powder	2%	.16 g
Sheep serum	10%	800 ul
Mab+Lev; Tween		8 ml
Final volume		8.8 ml

Heat at 65°C with frequent mixing. After the powder dissolves, cool at 4°C until needed.

Antibody solution

Anti-dig Ab	.1%	2.5 ul
Blocking Solution		2.5 ml
Final volume		2.5 ml

Use chilled blocking buffer. Store at 4°C until needed. Pre-absorb Ab in block solution for 1 hour before placing with embryos.

Protocol

Be careful during these washes. The embryos seem to be especially transparent and they are prone to float and stick.

1) Wash 3x 20 minutes with pre-warmed solution 1 at 65°C with rocking.
2) Wash 3x 20 minutes with pre-warmed solution 2 at 65°C with rocking.
3) Wash embryos 3x 5 minutes in Mab+Lev;Tween at room temperature with rocking.
4) Pre-block embryos in 5 ml Ab block solution for 2 hours at room temperature with rocking.
5) Replace with 2.5 ml Ab mixture. Rock gently overnight at 4° C.

Day 3 Post Antibody Washes

Solutions

Mab+Lev; Tween

Maleic Acid disodium salt	100 mM	2.4 g
NaCl	150 mM	1.315 g
Tween 20	.1%	150 ul

Levamisole	2 mM	.07224 g
ddH20		150 ml
Final volume		150 ml

Protocol

By Day 3 and the Mab washes, the embryos tend to sink and not stick to the sides of the vials.

1) Wash 3 x 5 minutes in Mab+Lev; Tween at room temp with rocking.
2) Wash 5 x 30-60 minutes in Mab+Lev; Tween at room temp with rocking.
3) Wash overnight in Mab+Lev; Tween at 4° C with rocking. Note, you can also wash at room temp for 2 hours with rocking and continue onto day 4.

Day 4 Alkaline Phosphatase Detection

Solutions

NTMT

NaCl	100 mM	600 ul 5M NaCL
Tris, pH 9.5	100 mM	3 ml 1M Tris
MgCl2	50 mM	1.5 ml 1M MgCl2
Tween 20	.1%	30 ul
Levamisole	2 mM	.0144 g
ddH2O		24.87 ml
Final volume		30 ml

Add the Levamisole and Tween 20 on the day of use.

Staining solution

Tween 20	.1%	2 ul
Levamisole	2 mM	.000996 g
BMPurple		2 ml
Final volume		2 ml

Protocol

1) Wash 3 x 10 minutes in NTMT at room temperature with rocking.
2) Replace NTMT with 1 ml of Staining solution.
3) Cover with aluminum foil and let stain for at room temperature with rocking.
4) Check for staining completion. In can be difficult to determine when the stains are done. As a rule of thumb, staining will take at least two hours, but you can continue staining until the background starts to come up. In general, a dissection microscope should be used to judge the staining intensity. With most probes, stain can proceed overnight at 4C with no problems. To speed the reaction, the solution can be replaces several times when you see a precipitate forming. Samples that will be sectioned will need to be over stained.
5) Rinse 2 x 5 minutes in PBT when staining is judged complete.
6) Post-fix in 4% paraformaldehyde for 1 hour at room temperature or overnight at 4°C.
7) Wash 2 x in PBT. If proceeding to gelatin embedding, proceed as normal. For storage or paraffin section, dehydrate in methanol series and store.

Stock Solutions

The following stock solutions are all computed for a final volume of 100 ml.

Depc H20

Depc	.1	100 ul
ddH20		100 ml
Final volume		100.1 ml

Add Depc and let the solution sit overnight. Autoclave the next day.

5M NaCl

NaCl	5 M	29.22 g
ddH20		100 ml
Final volume		100 ml

Mix well and autoclave.

1M Tris, pH 9.5

Tris (base)	1 M	12.11 g
ddH20	100 ml	
Final volume	100 ml	

Mix well. The pH will initially be around 11. Add hydrochloric acid to reduce pH to 9.5.

1M MgCl2

MgCl2	1 M	20.33 g
DdH20		100 ml
Final volume		100 ml

Mix well and autoclave.

20x SSC pH 4.5

NaCl	3 M	17.5 g
NaCitrate	300 mM	8.82 g
DdH20		100 ml
	OR	
Depc-H20		100 ml
Final volume		100 ml

pH with Citric acid to pH 4.5. If this is to be used for the Pre-hyb mix, use Depc-H$_2$0.

10% SDS

SDS	10%	10 g
ddH20		100 ml
Final volume		100 ml

Mix well and autoclave.

Chemicals

Anti-dig Ab	Boehringer	BM 1093 274
Blocking Solution	Boehringer	BM 1096 176
Formamide	Fisher	BP227-500
Heparin	Sigma	H8514
Maleic Acid disodium salt	Sigma	M9009
yeast RNA	Boehringer	109 495

Electrode Construction

Strip both ends of the 16 gauge wire and solder on banana plugs. Thread the other end of the 16 gauge wire though a holder. Put the platinum wire into the middle of

the copper strands and solder. The 16 gauge wire works well since the plastic coating forces the electrodes to be about 4 mm apart. Use a continuity meter to check that the connection is solid and that there isn't cross talk between the red and black sides. Apply non-conducting epoxy to the end of the electrodes. Make sure that there is enough to cover the joint between the platinum wire and the speaker wire.

Appendix C: Model Source Code

What follows below is the complete C source code for the model. The source code can be found on the CD-ROM as well.

main.c

```
/*************************************************************************
        This stochastic reaction-diffusion code is designed to study the
        problem of the binding of Retinoic acid binds to the retinoic acid
        receptors and the subsequent creation of the early members of the
        hox family: HoxA1, HoxA2, HoxB1 and Krox20

        Retinoic acid is assumed to be produced at a "point-source"
        located at the caudal section of the hindbrain and its distribution
        is determined by diffusion.

        This code requires Hox.h as its header file. 'Hox.c' furnishes
        all the routines that implement the physical effects of the reactions
        used as well as the diffusion code for RA,

        These functions are accessed by the Reaction[]() and Diffusion[]()
        functions, which are implemented as function arrays; this makes the
        bookeeping quite simple.

        Usage: a.out seed [val] stop [val] write [val]

*************************************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/times.h>
#include <time.h>
#include "Hox.h"
#include "header.h"
#include "UpdateAmu.h"
#include "ranlib.h"


char reactiontypes[NUM_FUNCS][25] = {
        "MakeRA",                   /* 0 */
        "BindRA",                   /* 1 */
        "DecayRA",                  /* 2 */
        "MakeRAR",                  /* 3 */
        "DecayRAR",                 /* 4 */
        "UnbindBRAR",               /* 5 */
        "DecayBRAR",                /* 6 */
        "ActivateA1",               /* 7 */
        "UnActivateA1",             /* 8 */
        "TranscribeA1",             /* 9 */
        "DecaymA1",                 /* 10 */
        "TranslateA1",              /* 11 */
        "Decaya1",                  /* 12 */
        "ActivateB1",               /* 13 */
        "UnActivateB1",             /* 14 */
        "TranscribeB1",             /* 15 */
        "DecaymB1",                 /* 16 */
        "TranslateB1",              /* 17 */
        "SuperActivateB1",          /* 18 */
        "UnSuperActivateB1",        /* 19 */
        "TranscribeSuperB1",        /* 20 */
        "RepressB1",                /* 21 */
        "UnRepressB1",              /* 22 */
        "AutoActivateB1",           /* 23 */
        "UnAutoActivateB1",          /* 24 */
        "TranscribeAutoB1",          /* 25 */
        "Decayb1",                  /* 26 */
        "ActivateB2",               /* 27 */
        "UnActivatedB2",            /* 28 */
```

```
        "TranscribeB2",         /* 29 */
        "DecaymB2",             /* 30 */
        "TranslateB2",          /* 31 */
        "Decayb2",              /* 32 */
        "Divide",               /* 33 */
        "ActivateKrox",         /* 34 */
        "UnActivateKrox",       /* 35 */
        "TranscribeKrox",       /* 36 */
        "DecaymKrox",           /* 37 */
        "RepressKrox",          /* 38 */
        "UnRepressKrox",        /* 39 */
        "TranslateKrox",        /* 40 */
        "Decaykrox",            /* 41 */
        "BindRXR",              /* 42 */
        "MakeRXR",              /* 43 */
        "DecayRXR",             /* 44 */
        "UnbindBRXR",           /* 45 */
         "DecayBRXR",           /* 46 */
         "Dimerize",            /* 47 */
        "UnDimerize",           /* 48 */
        "DecayDimer",           /* 49 */
        "Complexa1",            /* 50 */
        "Uncomplexa1",          /* 51 */
        "Decaya1Complex",       /* 52 */
        "Complexb1",            /* 53 */
        "Uncomplexb1",          /* 54 */
        "Decayb1Complex",       /* 55 */
        "MakeComplex",          /* 56 */
        "DecayComplex"};        /* 57 */


int
main(int argc,char * argv[])
{
        extern int DEBUG;
        void    srand48();
        double drand48();
        int     i,k,mu,x_i,switch_flag;
        struct stat buf;
        int     write_flag,count;
        int     diff_count,reac_count, result;
        int    diffusions, reactions, failures;
        long    seedval;
        float   a_summ,T,delta_t,t_stop,t_write;
        float   a0,r2a0;
        double  r1,r2;
```

```
        float   *image,*dummy;
        float    prob;
        int      a,b;
        char    name_buff[80];
        clock_t time;
        long double  d_time;
        CELL *          voxel;
        int     channels[NUM_FUNCS];
        int       border;
        int     counter;
        char * input;
        char * output;
        float tmp;
        float ktmp;
        FILE    *COUNT;
        FILE    *BORDER;
        FILE *fp;

        time = clock();
        seedval = 13;
        t_stop = 5000.0;
        t_write = 1.0;
        delta_t = 10.0;
        result = 0;
        x_i = 0;
        DEBUG = 0;
        input = (char *) NULL;

/******************** Get setup data from command line ********************/

        output = strdup("output");
        switch (argc) {
                case 1:
                        break;
                case 2:
                        input = strdup(argv[1]);
                        fp = fopen(input,"r");
                        seedval = (long) getValue(fp,"seed:");
                        t_stop = (float) getValue(fp,"stop:");
                        delta_t = (float) getValue(fp,"delta_t:");
                        fclose(fp);
                        break;
                case 3:
                        input = strdup(argv[1]);
                        output = strdup(argv[2]);
                        fp = fopen(input,"r");
```

```
                seedval = (long) getValue(fp,"seed:");
                t_stop = (float) getValue(fp,"stop:");
                delta_t = (float) getValue(fp,"delta_t:");
                fclose(fp);
                break;
        case 7:
                if (strcmp(argv[1],"seed") == 0) {
                        seedval = (double)atof(argv[2]);
                        t_stop = atof(argv[4]);
                        delta_t = atof(argv[6]);
                }
                else {
                        printf("Syntax error....\n");
                        exit (1);
                }
                break;
        default:
                printf("Syntax error...\n");
                exit (1);
                break;
}

/********************* Setup Initial Conditions ***************************/

/* Check that the output directory exists */
        sprintf(name_buff,"%s.%li/",output,seedval);
        if(stat(name_buff,&buf) == -1) {
                printf("Directory %s doesn't exist\n",name_buff);
                mkdir(name_buff,511);
        }
        else {
                printf("Directory %s exists\n",name_buff);
        }

        if ((COUNT = fopen("count","w")) == NULL) {
                printf("Cannot open data file count\n");
                exit (1);
        }

        sprintf(name_buff,"%s.%li/border",output,seedval);
        if ((BORDER = fopen(name_buff,"w")) == NULL) {
                printf("Cannot open data file count\n");
                exit (1);
        }

        srand48(seedval);
```

```
/*      Setup Initial Conditions for "Concentrations"        */

        NX = 40;
        dummy = (float *) malloc(2*NX*sizeof(float));
        image = (float *) malloc(2*NX*sizeof(float));
        voxel = init(2*NX);
        read_inputs(input, &initial_ra, &initial_rar, &D_ra,
             &a1hill, &b1hill, &b1auto,&rephill,&b2hill,C_mu,K);
/* Zero out some of the late events */

        tmp = C_mu[repressB1];
        ktmp = C_mu[activateKrox];
        C_mu[activateKrox] = 0.0;
        C_mu[repressB1] = 0.0;

        border = 19;
        prob = .01;
        for (i = 0; i < NX; i++) {
                a = (int) ignbin((long) initial_rar,prob);
                b = (int) ignbin((long) 10,.5);
                if(b <= 5)
                        voxel[i].rar = initial_rar+a;
                else
                        voxel[i].rar = initial_rar-a;
                a = (int) ignbin((long) initial_rar,prob);
                b = (int) ignbin((long) 10,.5);
                if(b <= 5)
                        voxel[i].rxr = initial_rar+a;
                else
                        voxel[i].rxr = initial_rar-a;

                voxel[i].plex = initial_rar/4;

/* genes */
                voxel[i].A1 = voxel[i].B2 = voxel[i].B1 = voxel[i].Krox = 1;
                dummy[i] = (float)i;

/* initial rhombomere identities */
                if(i < 20) {
                        voxel[i].id = R5;
                }
                else if ((i >= 20) && (i < 40)) {
                        voxel[i].id = R4;
                }
                else {
                        voxel[i].id = R3;
```

```
                }

                for (k = 0; k < NUM_FUNCS; k++) {
                        voxel[i].a_mu[k] = 0.0;
                }

/* Divide */
        voxel[i].a_mu[divide] = C_mu[divide];
        voxel[i].a_mu[makeRAR] = C_mu[makeRAR];
        voxel[i].a_mu[makeRXR] = C_mu[makeRXR];
        voxel[i].a_mu[decayRAR] = C_mu[decayRAR]*initial_rar;
        voxel[i].a_mu[makeComplex] = C_mu[makeComplex];
        voxel[i].a_mu[decayComplex] = C_mu[decayComplex];
           }
           for (k = 0; k < NUM_FUNCS; k++) {
                channels[k] = 0;
           }

/* Setup RA Source */

           counter = 0;
           voxel[0].ra = initial_ra;
           voxel[0].d_ra = voxel[0].ra*D_ra;
           update_cmu0(voxel,0.0);
           Update[bindRAR](voxel);
           Update[bindRXR](voxel);
           Update[decayRA](voxel);

           voxel[0].a_mu[divide] = 0.0;


/*****************************************************************/

           T = 0.0;
           write_flag = 0;
           count = 0;
           diff_count = 0;
           reac_count = 0;
           reactions = diffusions = failures = 0;


           while (T < t_stop) {                 /* start main loop */
                a0 = 0.0;
                for (i = 0; i < NX; i++) {   /* compute sum of react/diff params */
                        for (k = 0; k < NUM_FUNCS; k++) {
                                a0 += (voxel+i)->a_mu[k];    /* reaction values */
```

```
                        }
                        a0 += (voxel+i)->d_ra;
                }
                if (a0 == 0) {
                        printf("Unknown Error: a0 = 0...\n");
                        exit(1);
                }
                r1 = drand48();
                r2 = drand48();
                T += -log(r1)/a0;
/****          Check T to see if it is time to write data files       ****/
                if (T >= t_write) {

/* Start the late events  */
if(T >= 21000.5 && T <= 21500.5) {
        C_mu[repressB1] = tmp;
C_mu[activateKrox] = ktmp;
}



                        printf("Write Image Data.  Sim Time = %4.2e secs\n",T);
                        sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"ra",count);
                        for (i = 0; i < NX; i++) {
                                image[i] = voxel[i].ra;
                        }
                        write_gnu_data_file(image,dummy,NX,name_buff,2);

        sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"brar",count);
                        for (i = 0; i < NX; i++) {
                                image[i] = voxel[i].brar;
                        }
                        write_gnu_data_file(image,dummy,NX,name_buff,2);

        sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"brxr",count);
                        for (i = 0; i < NX; i++) {
                                image[i] = voxel[i].brxr;
                        }
                        write_gnu_data_file(image,dummy,NX,name_buff,2);

        sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"hoxa1",count);
                        for (i = 0; i < NX; i++) {
                                image[i] = voxel[i].a1;
                        }
                        write_gnu_data_file(image,dummy,NX,name_buff,2);
```

```
sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"thoxa1",count);
                for (i = 0; i < NX; i++) {
                        image[i] = voxel[i].a1 + voxel[i].a1plex;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"rar",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].rar;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"rxr",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].rxr;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"dimer",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].dimer;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"hoxb1",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].b1;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"plex",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].plex;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"a1plex",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].a1plex;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"b1plex",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].b1plex;
```

```
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"thoxb1",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].b1plex + voxel[i].b1;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"hoxb2",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].b2;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"krox20",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].krox;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"mhoxa1",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].mA1;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"mhoxb1",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].mB1;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"mhoxb2",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].mB2;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);

sprintf(name_buff,"%s.%li/%s.%d.dat",output,seedval,"mkrox20",count);
                for (i = 0; i < NX; i++) {
        image[i] = voxel[i].mKrox;
                }
                write_gnu_data_file(image,dummy,NX,name_buff,2);
                t_write += delta_t;
                d_time = (clock()-time)/CLOCKS_PER_SEC;
```

```
                        printf("Reac Count = %d  Diff Count =
%d\n",reac_count,diff_count);
                        printf("Incremental CPU Time = %4.2Le secs \n\n",d_time);
                        count++;
                        if(reac_count > 400000) {
                                printf("I'm stuck! Bailing out\n");
                                exit(1);
                        }
                        reac_count = 0;
                        diff_count = 0;
                        time = d_time;
                        for (k = 0; k < NUM_FUNCS; k++) {
                                fprintf(COUNT,"a_mu[%s (%d)] called %d times\n",
                                        reactiontypes[k],k,channels[k]);
                        }
                        fprintf(COUNT,
                                        "\nReac Count = %d Diff Count = %d  Fail Count =
%d\n",
                                        reactions,diffusions,failures);
                        fflush(COUNT);

                        fprintf(BORDER,"%d\n",border);
                        fflush(BORDER);
                }
                r2a0 = r2*a0;
                a_summ = 0;
                mu = 0;
                switch_flag = 0;
                for (i = 0; i < NX; i++) {
                        x_i = i;
                        for (k = 0; k < NUM_FUNCS; k++) {
                                mu = k;
                                a_summ += voxel[i].a_mu[k];
                                if (a_summ >= r2a0) {
                                        switch_flag = 1;
                                        goto React;
                                }
                        }
                        a_summ += voxel[i].d_ra;
                        if (a_summ >= r2a0) {
                                switch_flag = 2;
                                mu = 0;
                                goto React;
                        }
                }
React:  if (switch_flag != 0) {
```

```
                        switch (switch_flag) {
                                case 1:
if(DEBUG) {
        printf("calling a_mu[%d] in cell %d\n",mu,x_i);
}
                                        channels[mu] += 1;
                                        if(mu == divide) {
                                                (voxel+x_i)->a_mu[divide] = 0.0;
                                                printf("Cell %d is dividing!\n",x_i);
                                                if(x_i <=  border)
                                                        border++;
                                                NX++;
                                                result = Reaction[mu](voxel+x_i,NX);
                                voxel[x_i].a_mu[divide] = 0.0;
                                voxel[x_i+1].a_mu[divide] = 0.0;
                                        }
                                        else {
                                                result = Reaction[mu](voxel+x_i);
                                        }
                                        reac_count += 1;
                                        reactions +=1;
                                break;
                                case 2:
if(DEBUG) {
        printf("calling diffusion in cell %d\n",x_i);
}
                                        result = Diffusion[mu](voxel+x_i);
                                        diff_count += 1;
                                        diffusions +=1;
                                        break;
                        }
                        update_cmu0(voxel,T);
                }
                if(!result) {
/* Back out the time */
                        T -= -log(r1)/a0;
                        if(switch_flag == 1) {
                                printf("Error: Called a_mu[%s (%d)] = %f in cell %d\n",
                                        reactiontypes[mu],mu,voxel[x_i].a_mu[mu],x_i);
                        }
                        failures += 1;
                        exit(1);
                }
        }           /* End Main Loop */
```

```
        printf("Reac Count = %d  Diff Count = %d, Fail Count = %d\n",
        reactions,diffusions,failures);
        fclose(COUNT);
        fclose(BORDER);
        exit (0);
}
```

inputs.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "header.h"


double
getValue(FILE * fp,char * tag) {
  char buf[80];
  char *ptr;
  char *res;
  double val;

  res = fgets(buf,80,fp);

/* Look for the line with the right tag */
  while(!(ptr = strstr(buf,tag)) && res) {
    res = fgets(buf,80,fp);
  }
  if(!res) {
   printf("The tag %s was not found\n",tag);
   exit(1);
  }

/* Now that we are on the right line, look for the colon */

  while(*ptr != ':') {
    ptr++;
  }

/* Move past the colon */
  ptr++;

/* the next thing is the value we want. */
  val = atof(ptr);

/* Rewind the stream to the beginning of the file */
```

```
  rewind(fp);

  return val;
}

void
read_inputs(char * filename,int * init_ra, int * init_rar, float *D_ra, double *a1hill,
        double *b1hill, double *b1auto, double *rephill, double *b2hill, float c_mu[],
        float K[])
{

  FILE * fp;
  char buf[20];
  int i;

  fp = fopen(filename,"r");

  *init_ra = (int) getValue(fp,"initial_ra");
  *init_rar = (int) getValue(fp,"initial_rar");

  *D_ra   = (float) getValue(fp,"D_ra");
  *a1hill = (double) getValue(fp,"a1hill");
  *b1hill = (double) getValue(fp,"b1hill");
  *b1auto = (double) getValue(fp,"b1auto");
  *rephill = (double) getValue(fp,"rephill");
  *b2hill = (double) getValue(fp,"b2hill");

// read in the production rate values
  for(i = 0; i < NUM_FUNCS; i++) {
     sprintf(buf,"c_mu%d",i);
     c_mu[i] = (float) getValue(fp,buf);
  }
  for(i = 0; i < 7; i++) {
     sprintf(buf,"K%d",i);
     K[i] = (float) getValue(fp,buf);
  }
}
```

II.c

```
#include <stdio.h>
#include <strings.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "ranlib.h"
#include "Hox.h"
#include "header.h"
#include "UpdateAmu.h"


void
update(CELL *c) {
   int i;
   for(i = 0; i < NUM_FUNCS; i++) {
      Update[i](c);
   }
}

void
print_cell(FILE * DATA,CELL * c) {
        if(c->num == 0) {
                fprintf(DATA,"num\tra\trar\trxr\tbrar\tbrxr\tdimer\t");
                fprintf(DATA,"a1\tb1\tb2\tkrox\tmA1\tmB1\tmB2\t");
                fprintf(DATA,"mKrox\tplex\ta1plex\tb1plex\n");
        }
        fprintf(DATA,"%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t
%d\t%d\t%d\t%d\n",c->num+1, c->ra, c->rar, c->rxr, c->brar, c->brxr, c->dimer,
                        c->a1, c->b1, c->b2, c->krox, c->mA1, c->mB1, c->mB2,
                        c->mKrox, c->plex, c->a1plex, c->b1plex);
}


void
divide_resources(CELL * mom, CELL * daughter) {
    int a;

    float prob = .5;
    a = (int) ignbin((long) mom->a1 ,prob);
    daughter->a1 -= a;
    mom->a1 = a;

    a = (int) ignbin((long) mom->mA1 ,prob);
    daughter->mA1 -= a;
    mom->mA1 = a;

    a = (int) ignbin((long) mom->b1 ,prob);
    daughter->b1 -= a;
    mom->b1 = a;

    a = (int) ignbin((long) mom->mB1 ,prob);
```

```
daughter->mB1 -= a;
mom->mB1 = a;

a = (int) ignbin((long) mom->b2 ,prob);
daughter->b2 -= a;
mom->b2 = a;

a = (int) ignbin((long) mom->mB2 ,prob);
daughter->mB2 -= a;
mom->mB2 = a;

a = (int) ignbin((long) mom->mKrox ,prob);
daughter->mKrox -= a;
mom->mKrox = a;

a = (int) ignbin((long) mom->krox ,prob);
daughter->krox -= a;
mom->krox = a;

a = (int) ignbin((long) mom->brar ,prob);
daughter->brar -= a;
mom->brar = a;

a = (int) ignbin((long) mom->brxr ,prob);
daughter->brxr -= a;
mom->brxr = a;

a = (int) ignbin((long) mom->dimer ,prob);
daughter->dimer -= a;
mom->dimer = a;

a = (int) ignbin((long) mom->rar ,prob);
daughter->rar -= a;
mom->rar = a;

a = (int) ignbin((long) mom->plex ,prob);
daughter->plex -= a;
mom->plex = a;

a = (int) ignbin((long) mom->a1plex ,prob);
daughter->a1plex -= a;
mom->a1plex = a;

a = (int) ignbin((long) mom->b1plex ,prob);
daughter->b1plex -= a;
mom->b1plex = a;
```

```
    a = (int) ignbin((long) mom->rxr ,prob);
    daughter->rxr -= a;
    mom->rxr = a;
#ifdef DEBUG
    printf("mom has %d, daughter has %d rar\n",a,daughter->rar);
#endif
}

void
add(CELL *head, CELL *new)
{
   CELL *ptr;
   ptr = head;
   while((ptr->next != (CELL*) NULL) && (new->num > ptr->next->num)) {
        ptr = ptr->next;
   }
   if(ptr->next == (CELL*) NULL) {
     ptr->next = new;
     new->next =  (CELL *) NULL;
   }
   else {
     new->next = ptr->next;
     ptr->next = new;
   }
}

int
Divide(CELL *afterme, int NX)
{
   CELL *ptr;
   int i;
   int ncells;

   int flag = 0;
   ptr = afterme;
   ncells = NX-afterme->num;
   for(i = 0; i < ncells; i++) {
      (afterme+i)->num++;
   }
   memmove(afterme+1,afterme, ncells*sizeof(CELL));
   memcpy(afterme,afterme+1,sizeof(CELL));
   divide_resources(afterme,afterme+1);

   update(afterme);
   update(afterme+1);
```

```
   afterme->num--;
   flag = 1;
   return flag;
}

CELL*
init(int size) {
   int i;
   CELL * head;
   head = (CELL*) calloc(size,sizeof(CELL));
   for(i = 0; i < size-1; i++) {
     (head+i)->num = i;
     (head+i)->next = (head+i+1);
   }
   (head+(size-1))->num = size-1;
   (head+(size-1))->next = (CELL *) NULL;
   return head;
}
```

write_gnu_data_file.c

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <malloc.h>

void
write_gnu_data_file(float *array,float *farray,int length,char*
fname,int d_flag)
{
       int     jj;
       float   norm_dist;
       FILE    *fpo;

       if ((fpo = fopen(fname,"w")) == NULL) {
         printf("Cannot open gnu data file %s\n",fname);
         exit (0);
       }

       for (jj = 0; jj < length; jj++) {
         if (d_flag == 1)
            norm_dist = (float)jj/(float)length;
          else if (d_flag == 2)
              norm_dist = (float)jj+1.0;
         else
            norm_dist = farray[jj];
```

```
        fprintf(fpo,"%0.3f    %0.5f\n",norm_dist,array[jj]);
      }
        fprintf(fpo,"\n");
        fclose(fpo);
}
```

header.h

```
#ifndef __HEADER_H
#define __HEADER_H
#include "Hox.h"

void update();
void divide_resources();
void print_cell();
void add();
int Divide();
CELL* init();
double getValue();
void read_inputs();
void write_gnu_data_file();

#endif
```

Hox.h

```
/*
 *      Header file for stochastic simulation of study of Retinoic Acid
 *      diffusion and the production of the early members of the hox
 *      family using the extended Gillespie formulation for 1-dimensional
 *      reaction-diffusion.
 *
 */

#ifndef __HOX_H_
#define __HOX_H_

#defineNUM_FUNCS 58
#define PERMS 0666

typedef enum { R3,R4,R5 } rhombomere;
typedef enum {A,B} Repressor;
```

```
/* Note that the convention adopted is that the uppercase letters stand
 * for the DNA and the lowercase stand for proteins
 */

typedef struct cell {
      int  num;                 /* Cell number */
      rhombomere  id;           /* The identity of the cell */
      int  ra;                  /* number of unbound RA molecules */
      int  rar;                 /* number of RA receptors */
      int  rxr;                 /* number of RA receptors */
      int  brar;                /* number of bound RA molecules  */
      int  brxr;                /* number of bound RA molecules  */
      int  dimer;               /* number of rar/rxr dimers */
      int  A1;                  /* number of A1 genes */
      int  actA1;               /* number of activated A1 genes */
      int  B1;                  /* number of B1 genes */
      int  actB1;               /* number of activated B1 genes */
      int  superactB1;          /* number of super activated B1 genes */
      int  repB1;               /* number of repressed B1 genes */
      int  autoB1;              /* number of auto activated B1 genes */
      int  B2;                  /* number of B2 genes */
      int  actB2;               /* number of activated B2 genes */
      int  Krox;                /* number of Krox20 genes */
      int  actKrox;             /* number of activated Krox20 genes */
      int  repKrox;             /* number of repressed Krox20 genes */
      Repressorkrep;            /* what the current repressor for krox is */
      int  plex;                /* number of complexes availble */
      int  mA1;                 /* number of A1 mRNA */
      int  mB1;                 /* number of B1 mRNA */
      int  mB2;                 /* number of B2 mRNA */
      int  mKrox;               /* number of Krox20 mRNA */
      int  a1;                  /* number of a1 proteins */
      int  a1plex;              /* number of a1+pbx+prep molecules */
      int  b1;                  /* number of b1 proteins */
      int  b1plex;              /* number of a1+pbx+prep molecules */
      int  b2;                  /* number of b2 proteins */
      int  krox;                /* number of krox20 proteins */
      float   d_ra;             /* Retinoic acid diffusion coefficient */
      float   a_mu[NUM_FUNCS];/* Reaction probabilities */
      struct cell *next; /* Pointer to the next cell */
} CELL;


enum {makeRA,               /* 0 */
   bindRAR,                 /* 1 */
   decayRA,                 /* 2 */
```

```
makeRAR,            /* 3 */
decayRAR,           /* 4 */
unbindBRAR,         /* 5 */
decayBRAR,          /* 6 */
activateA1,         /* 7 */
unActivateA1,       /* 8 */
transcribeA1,       /* 9 */
decaymA1,           /* 10 */
translateA1,        /* 11 */
decaya1,            /* 12 */
activateB1,         /* 13 */
unActivateB1,       /* 14 */
transcribeB1,       /* 15 */
decaymB1,           /* 16 */
translateB1,        /* 17 */
superActivateB1,    /* 18 */
unSuperActivateB1,  /* 19 */
transcribeSuperB1,  /* 20 */
repressB1,          /* 21 */
unRepressB1,        /* 22 */
autoActivateB1,     /* 23 */
unAutoActivateB1,   /* 24 */
transcribeAutoB1,   /* 25 */
decayb1,            /* 26 */
activateB2,         /* 27 */
unActivateB2,       /* 28 */
transcribeB2,       /* 29 */
decaymB2,           /* 30 */
translateB2,        /* 31 */
decayb2,            /* 32 */
divide,             /* 33 */
activateKrox,       /* 34 */
unActivateKrox,     /* 35 */
transcribeKrox,     /* 36 */
decaymKrox,         /* 37 */
repressKrox,        /* 38 */
unRepressKrox,      /* 39 */
translateKrox,      /* 40 */
decaykrox,          /* 41 */
bindRXR,            /* 42 */
makeRXR,            /* 43 */
decayRXR,           /* 44 */
unbindBRXR,         /* 45 */
decayBRXR,          /* 46 */
dimerize,           /* 47 */
unDimerize,         /* 48 */
```

```
    decayDimer,             /* 49 */
    complexa1,              /* 50 */
    unComplexa1,            /* 51 */
    decaya1Complex,         /* 52 */
    complexb1,              /* 53 */
    unComplexb1,            /* 54 */
    decayb1Complex,         /* 55 */
    makeComplex,            /* 56 */
    decayComplex};          /* 57 */


typedef int (*REACTION)();

/******************** Function Declarations ***************************/

int     update_cmu0(CELL*,float);

int     MakeRA(CELL *);             /* a_mu[0] */
int     BindRAR(CELL *);            /* a_mu[1] */
int     DecayRA(CELL *);            /* a_mu[2] */
int     MakeRAR(CELL *);            /* a_mu[3] */
int     DecayRAR(CELL *);           /* a_mu[4] */
int     UnbindBRAR(CELL *);         /* a_mu[5] */
int     DecayBRAR(CELL *);          /* a_mu[6] */
int     ActivateA1(CELL *);         /* a_mu[7] */
int     UnActivateA1(CELL *);       /* a_mu[8] */
int     TranscribeA1(CELL *);       /* a_mu[9] */
int     DecaymA1(CELL *);           /* a_mu[10] */
int     TranslateA1(CELL *);        /* a_mu[11] */
int     Decaya1(CELL *);            /* a_mu[12] */
int     ActivateB1(CELL *);         /* a_mu[13] */
int     UnActivateB1(CELL *);       /* a_mu[14] */
int     TranscribeB1(CELL *);       /* a_mu[15] */
int     DecaymB1(CELL *);           /* a_mu[16] */
int     TranslateB1(CELL *);        /* a_mu[17] */
int     SuperActivateB1(CELL *);    /* a_mu[18] */
int     UnSuperActivateB1(CELL *);/* a_mu[19] */
int     TranscribeSuperB1(CELL *);/* a_mu[20] */
int     RepressB1(CELL *);          /* a_mu[21] */
int     UnRepressB1(CELL *);        /* a_mu[22] */
int     AutoActivateB1(CELL *);     /* a_mu[23] */
int     UnAutoActivateB1(CELL *);/* a_mu[24] */
int     TranscribeAutoB1(CELL *);/* a_mu[25] */
int     Decayb1(CELL *);            /* a_mu[26] */
int     ActivateB2(CELL *);         /* a_mu[27] */
int     UnActivateB2(CELL *);       /* a_mu[28] */
```

```
int     TranscribeB2(CELL *);       /* a_mu[29] */
int     DecaymB2(CELL *);           /* a_mu[30] */
int     TranslateB2(CELL *);        /* a_mu[31] */
int     Decayb2(CELL *);            /* a_mu[32] */
int     Divide(CELL *, int);        /* a_mu[33] */
int     ActivateKrox(CELL *);       /* a_mu[34] */
int     UnActivateKrox(CELL *);     /* a_mu[35] */
int     TranscribeKrox(CELL *);     /* a_mu[36] */
int     DecaymKrox(CELL *);         /* a_mu[37] */
int     RepressKrox(CELL *);        /* a_mu[38] */
int     UnRepressKrox(CELL *);      /* a_mu[39] */
int     TranslateKrox(CELL *);      /* a_mu[40] */
int     Decaykrox(CELL *);          /* a_mu[41] */
int     BindRXR(CELL *);            /* a_mu[42] */
int     MakeRXR(CELL *);            /* a_mu[43] */
int     DecayRXR(CELL *);           /* a_mu[44] */
int     UnbindBRXR(CELL *);         /* a_mu[45] */
int     DecayBRXR(CELL *);          /* a_mu[46] */
int     Dimerize(CELL *);           /* a_mu[47] */
int     UnDimerize(CELL *);         /* a_mu[48] */
int     DecayDimer(CELL *);         /* a_mu[49] */
int     Complexa1(CELL *);          /* a_mu[50] */
int     Uncomplexa1(CELL *);        /* a_mu[51] */
int     Decaya1Complex(CELL *);     /* a_mu[52] */
int     Complexb1(CELL *);          /* a_mu[53] */
int     Uncomplexb1(CELL *);        /* a_mu[54] */
int     Decayb1Complex(CELL *);     /* a_mu[55] */
int     MakeComplex(CELL *);        /* a_mu[56] */
int     DecayComplex(CELL *);       /* a_mu[57] */

int     RA_Diffusion(CELL *);
/********************** Initializations *********************************/


static REACTION     Reaction[] = {
                        MakeRA,                 /* 0 */
                        BindRAR,                /* 1 */
                        DecayRA,                /* 2 */
                        MakeRAR,                /* 3 */
                        DecayRAR,               /* 4 */
                        UnbindBRAR,             /* 5 */
                        DecayBRAR,              /* 6 */
                        ActivateA1,             /* 7 */
                        UnActivateA1,           /* 8 */
                        TranscribeA1,           /* 9 */
                        DecaymA1,               /* 10 */
```

```
TranslateA1,            /* 11 */
Decaya1,                /* 12 */
ActivateB1,             /* 13 */
UnActivateB1,           /* 14 */
TranscribeB1,           /* 15 */
DecaymB1,               /* 16 */
TranslateB1,            /* 17 */
SuperActivateB1,        /* 18 */
UnSuperActivateB1,      /* 19 */
TranscribeSuperB1,      /* 20 */
RepressB1,              /* 21 */
UnRepressB1,            /* 22 */
AutoActivateB1,         /* 23 */
UnAutoActivateB1,       /* 24 */
TranscribeAutoB1,       /* 25 */
Decayb1,                /* 26 */
ActivateB2,             /* 27 */
UnActivateB2,           /* 28 */
TranscribeB2,           /* 29 */
DecaymB2,               /* 30 */
TranslateB2,            /* 31 */
Decayb2,                /* 32 */
Divide,                 /* 33 */
ActivateKrox,           /* 34 */
UnActivateKrox,         /* 35 */
TranscribeKrox,         /* 36 */
DecaymKrox,             /* 37 */
RepressKrox,            /* 38 */
UnRepressKrox,          /* 39 */
TranslateKrox,          /* 40 */
Decaykrox,              /* 41 */
BindRXR,                /* 42 */
MakeRXR,                /* 43 */
DecayRXR,               /* 44 */
UnbindBRXR,             /* 45 */
DecayBRXR,              /* 46 */
Dimerize,               /* 47 */
UnDimerize,             /* 48 */
DecayDimer,             /* 49 */
Complexa1,              /* 50 */
Uncomplexa1,            /* 51 */
Decaya1Complex,         /* 52 */
Complexb1,              /* 53 */
Uncomplexb1,            /* 54 */
Decayb1Complex,         /* 55 */
MakeComplex,            /* 56 */
```

```
                        DecayComplex};                  /* 57 */


static REACTION      Diffusion[] = { RA_Diffusion };

float    C_mu[NUM_FUNCS];
float K[7];

int      initial_ra;
int      initial_rar;
float    D_ra;
float    Kg;
double a1hill;
double b1hill;
double b1auto;
double  rephill;
double b2hill;
float    G1;
float    G2;
int NX;
int DEBUG;

#endif
```

Hox.c

```
/*************************************************************************
        This file contains the functions which implement the
        reaction channels for the  RA/Hox study; it also contains
        the function required to implement the diffusion components
        of Retinoic Acid.

        Note that the a_mu and d_mu values are updated during these
        function calls.  No updating of these quantities is done in
        the main program.


*************************************************************************/
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "Hox.h"
#include "UpdateAmu.h"
```

```
#include "ranlib.h"

int
update_cmu0(CELL *c, float T)
{
        int flag = 0;

        if(c->num != 0) {
                fprintf(stderr,"Error in update_cmu0:");
                fprintf(stderr," Trying to update in cell %d",c->num);
                goto cleanup;
        }
        c->a_mu[makeRA] = C_mu[makeRA]*T*exp(-.005*T);
        flag = 1;

cleanup:
        return flag;
}

int
MakeRA(CELL * c) /*   *-> ra a_mu[0] */
{
        int flag = 0;
        int affected = 3;
        int todo[3] = {bindRXR,bindRAR,decayRA};
        int i;
        c->ra += 1;
        c->d_ra = D_ra*(c->ra);

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        flag = 1;
        return flag;
}

int
BindRAR(CELL *c)    /* ra + rar -> brar a_mu[1] */
{
        int flag = 0;
        int affected = 7;
        int i;
        int todo[7]= {bindRAR,decayRA,bindRXR,decayRAR,unbindBRAR,
                                decayBRAR,dimerize};

        if(c->ra < 1 || c->rar < 1) {
```

```
                fprintf(stderr,"Error in BindRAR:");
                fprintf(stderr," Cell %d has %d ra and %d rar\n",c->num,c->ra,c->rar);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->ra -= 1;
        c->rar -= 1;
        c->brar += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        c->d_ra = D_ra*(c->ra);
        flag = 1;
cleanup:
        return flag;
}

int
DecayRA(CELL * c)             /*   ra->null  a_mu[2] */
{
        int flag = 0;
        int affected = 3;
        int todo[3] = {bindRAR,bindRXR,decayRA};
        int i;
        if(c->ra < 1) {
                fprintf(stderr,"Error in DecayRA:");
                fprintf(stderr," Cell %d has %d RA\n", c->num,c->ra);
                goto cleanup;
        }
        c->ra -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        c->d_ra = D_ra*(c->ra);
        flag = 1;

cleanup:
        return flag;
}

int
MakeRAR(CELL * c) /*   *-> rar a_mu[3] */
{
```

```
	int affected = 2;
	int todo[2] = {bindRAR,decayRAR};
	int flag = 0;
	int i;
	c->rar += 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}
	flag = 1;
	return flag;
}

int
DecayRAR(CELL * c)			/*   rar->null a_mu[4]*/
{
	int affected = 2;
	int todo[2] = {bindRAR,decayRAR};
	int flag = 0;
	int i;
	if(c->rar < 1) {
		fprintf(stderr,"Error in DecayRAR:");
		fprintf(stderr," Cell %d has %d RAR\n", c->num,c->rar);
		goto cleanup;
	}
	c->rar -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}
	flag = 1;

cleanup:
	return flag;
}

int
UnbindBRAR(CELL *c)			/* brar-> ra + rar -> a_mu[5] */
{
	int flag = 0;
	int affected = 7;
	int i;
	int todo[7] = {bindRAR,bindRXR,decayRA,decayRAR,unbindBRAR,
				decayBRAR,dimerize};
	if(c->brar < 1) {
		fprintf(stderr,"Error in UnbindBRA:");
		fprintf(stderr," Cell %d has %d brar.\n",c->num,c->brar);
		goto cleanup;
```

```
        }
/* Change the relevant quantities */
        c->ra += 1;
        c->rar += 1;
        c->brar -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        c->d_ra = D_ra*(c->ra);

        flag = 1;
cleanup:
        return flag;
}

int
DecayBRAR(CELL *c)                  /* brar-> null  a_mu[6] */
{
        int flag = 0;
        int affected = 3;
        int todo[3] = {unbindBRAR,decayBRAR,dimerize};
        int i;
        if(c->brar < 1) {
                fprintf(stderr,"Error in DecayBRA:");
                fprintf(stderr," Cell %d has %d brar.\n",c->num,c->brar);
                goto cleanup;
        }
/* Change the relevant quantities */
        c->brar -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
        cleanup:

        return flag;
}

int
ActivateA1(CELL * c)                /*  brar + A1 -> actA1  a_mu[7];*/
{
        int flag = 0;
        int affected = 7;
        int i;
```

```
        int todo[7] = {unDimerize,decayDimer,activateA1,transcribeA1,
                activateB1,unActivateA1,repressB1};

        if(c->A1 < 1 || c->dimer < 1 || c->actA1 > 1) {
                fprintf(stderr,"Error in ActivateA1:");
                fprintf(stderr," Cell %d has %d A1, %d actA1 and %d dimer.\n",
      c->num,c->A1,c->actA1,c->dimer);
                goto cleanup;
        }
        c->A1 = 0;
        c->actA1 = 1;
        c->dimer -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
}

int
UnActivateA1(CELL * c)                  /* actA1 -> A1,brar  a_mu[8]*/
{
        int flag = 0;
        int affected = 8;
        int todo[8] = {unDimerize,decayDimer,decayBRAR,activateA1,unActivateA1,
                        transcribeA1,activateB1,repressB1};
        int i;
        if(c->actA1 < 1) {
                fprintf(stderr,"Error in UnActivateA1:");
                fprintf(stderr," Cell %d has %d actA1\n", c->num,c->actA1);
                goto cleanup;
        }
        c->A1 = 1;
        c->actA1 = 0;
        c->dimer += 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
        cleanup:
        return flag;
```

```
}

int
TranscribeA1(CELL * c)                  /* actA1 -> mA1  a_mu[9]*/
{

        int flag = 0;
        int affected = 9;
        int todo[9] = {unDimerize,decayDimer,activateA1,unActivateA1,
                        transcribeA1,translateA1,activateB1,repressB1,
                        decaymA1};
        int i;

        if(c->A1 < 1) {
                fprintf(stderr,"Error in TranscribeA1:");
                fprintf(stderr," Cell %d has %d A1\n", c->num,c->A1);
                goto cleanup;
        }

        c->mA1 += 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
        cleanup:
        return flag;
}

int
DecaymA1(CELL * c)                      /* mA1 -> null  a_mu[10]*/
{

        int flag = 0;
        int affected = 3;
        int todo[3] = {transcribeA1,translateA1,decaymA1};
        int i;

        if(c->mA1 < 1) {
                fprintf(stderr,"Error in DecaymA1:");
                fprintf(stderr," Cell %d has %d mA1\n", c->num,c->mA1);
                goto cleanup;
        }

        c->mA1 -= 1;
        for(i = 0; i < affected; i++) {
```

```
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}

int
TranslateA1(CELL * c)                   /* mA1 -> a1 a_mu[11]*/
{
        int flag = 0;
        int affected = 4;
        int todo[4] = {translateA1,decaya1,decaymA1,complexa1};
        int i;
        if(c->mA1 < 1) {
                fprintf(stderr,"Error in TranslateA1:");
                fprintf(stderr," Cell %d has %d mA1\n", c->num,c->mA1);
                goto cleanup;
        }

        c->a1 += 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
                return flag;
}

int
Decaya1(CELL *c)                /* a1 -> null  a_mu[12]*/
{
        int flag = 0;
        int affected = 2;
        int i;
        int todo[2] = {decaya1,complexa1};
        if(c->a1 < 1) {
                fprintf(stderr,"Error in Decaya1:");
                fprintf(stderr," Cell %d has %d a1.\n",c->num,c->a1);
                goto cleanup;
        }
        c->a1 -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
```

```
	}

	flag = 1;
cleanup:
	return flag;
}


int
ActivateB1(CELL * c)			/*  brar + B1 -> actB1 a_mu[13]*/
{
	int i;
	int flag = 0;
	int affected = 11;
	int todo[11] = {unDimerize,decayDimer,activateB1,unActivateB1,
				activateA1,transcribeB1,superActivateB1,repressB1,
				autoActivateB1,activateB2,unActivateB2};

	if(c->B1 < 1 || c->dimer < 1) {
		fprintf(stderr,"Error in ActivateB1:");
		fprintf(stderr," Cell %d has %d B1 and %d dimer\n",
		c->num,c->B1,c->dimer);
		goto cleanup;
	}
	c->B1 = 0;
	c->dimer -= 1;
	c->actB1 = 1;

/* Occasionally, activate the b2 gene */
	i =  (int) ignbin((long) 10, .25);
	if(c->B2 == 1 && i <= 2) {
		c->actB2 = 1;
		c->B2 = 0;
	}
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;

cleanup:
	return flag;
}

int
UnActivateB1(CELL * c)			/*  actB1 -> B1,brar a_mu[14]*/
```

```
{
        int flag = 0;
        int affected = 9;
        int i;
        int todo[9] = {unDimerize,decayDimer,activateB1,unActivateB1,
                        transcribeB1,superActivateB1,repressB1,autoActivateB1,
                                activateA1};
        if(c->actB1 < 1) {
                fprintf(stderr,"Error in UnactivateB1:");
                fprintf(stderr," Cell %d has %d actB1.\n",
               c->num,c->actB1);
                goto cleanup;
        }
        c->B1 = 1;
        c->dimer += 1;
        c->actB1 = 0;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
}

int
TranscribeB1(CELL * c)                  /*  superactB1 -> mB1 a_mu[15]*/
{

        int flag = 0;
        int affected = 11;
        int i;
        int todo[11] = {unDimerize,decayDimer,activateB1,unActivateB1,
                        transcribeB1,superActivateB1,translateB1,repressB1,
                        autoActivateB1,decaymB1,activateA1};
        if(c->actB1 < 1) {
                fprintf(stderr,"Error in TranscribeB1:");
                fprintf(stderr," Cell %d has %d actB1.\n", c->num,c->actB1);
                goto cleanup;
        }
        c->mB1 += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
```

```
	flag = 1;

	cleanup:
	return flag;
	}

int
DecaymB1(CELL * c)			/* mA1 -> null  a_mu[16]*/
{

	int flag = 0;
	int affected = 3;
	int i;
	int todo[3] = {transcribeB1,translateB1,decaymB1};

	if(c->mB1 < 1) {
		fprintf(stderr,"Error in DecaymB1:");
		fprintf(stderr," Cell %d has %d mA1\n", c->num,c->mB1);
		goto cleanup;
	}

	c->mB1 -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
TranslateB1(CELL * c)			/*  mB1 -> b1 a_mu[17]*/
{
	int flag = 0;
	int affected = 5;
	int i;
	int todo[5] = {translateB1,complexb1,decaymB1,decayb1,repressKrox};
	if(c->mB1 < 1) {
		fprintf(stderr,"Error in TranslateB1:");
		fprintf(stderr," Cell %d has %d mB1\n", c->num,c->mB1);
		goto cleanup;
	}
	c->b1 += 1;
	for(i = 0; i < affected; i++) {
```

```
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
}


int
SuperActivateB1(CELL *c)            /* actB1+a1 ->superactB1 a_mu[18] */
{
        int flag = 0;
        int affected = 9;
        int i;
        int todo[9] = {unActivateB1,decaya1,superActivateB1,transcribeB1,
                                unSuperActivateB1,transcribeSuperB1,autoActivateB1,
                                        decaya1Complex,unComplexa1};
        if(c->actB1 < 1 || c->a1plex < 1) {
                fprintf(stderr,"Error in SuperActivateB1:");
                fprintf(stderr," Cell %d has %d actB1 and %d a1plex\n",
                        c->num,c->actB1,c->a1plex);
                goto cleanup;
        }
        c->actB1 = 0;
        c->superactB1 = 1;
        c->a1plex -= 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
}

int
UnSuperActivateB1(CELL *c)                  /*  superactB1 -> actB1, a1 a_mu[19]*/
{
        int flag = 0;
        int affected = 13;
        int i;
        int todo[13] = {decaya1,unActivateB1,superActivateB1,transcribeB1,
```

```
                                unSuperActivateB1,transcribeSuperB1,activateA1,

        decayDimer,unDimerize,repressB1,decaya1Complex,
                                        unComplexa1,repressB1};

        if(c->superactB1 < 1) {
                fprintf(stderr,"Error in UnSuperActivateB1:");
                fprintf(stderr," Cell %d has %d superactB1\n",c->num,c->superactB1);
                goto cleanup;
        }
//      c->actB1 = 1;
//      c->superactB1 = 0;
//      c->a1plex += 1;

        c->B1 = 1;
        c->superactB1 = 0;
        c->actB1 = 0;
        c->a1plex += 1;
        c->dimer += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;

}

int
TranscribeSuperB1(CELL * c)                     /*  superactB1 -> mB1 a_mu[20]*/
{
        int flag = 0;
        int affected = 14;
        int i;
        int todo[14] = {unDimerize,decayDimer,activateA1,activateB1,
                                superActivateB1,unSuperActivateB1,transcribeSuperB1,
                                translateB1,repressB1,autoActivateB1,decaymB1,
                                        repressKrox,unComplexa1,decayb1};
        if(c->superactB1 < 1) {
                fprintf(stderr,"Error in TranscribeSuperB1:");
                fprintf(stderr," Cell %d has %d actB1.\n", c->num,c->superactB1);
                goto cleanup;
        }
        c->mB1 += 2;            /* This should be changed to 5 or so */
```

```
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}
	flag = 1;

cleanup:
	return flag;
}

int
RepressB1(CELL * c)			/*  B1+brar -> repB1 a_mu[21]*/
{
	int flag = 0;
	int affected = 7;
	int i;
	int todo[7] = {unDimerize,decayDimer,activateA1,activateB1,
			unRepressB1,autoActivateB1,repressB1};
	if(c->B1 < 1 || c->dimer < 1) {
		fprintf(stderr,"Error in RepressB1:");
		fprintf(stderr," Cell %d has %d B1 and %d dimer\n",
						c->num,c->B1,c->dimer);
		goto cleanup;
	}
	c->repB1 = 1;
	c->B1 = 0;
	c->dimer -= 1;

	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;

cleanup:
	return flag;
}

int
UnRepressB1(CELL * c)			/*  B1 -> repB1,brar  a_mu[22]*/
{
	int flag = 0;
	int affected = 7;
	int i;
	int todo[7] = {unDimerize,decayDimer,activateA1,activateB1,
			unRepressB1,autoActivateB1,repressB1};
	if(c->repB1 < 1) {
```

```
                fprintf(stderr,"Error in UnRepressB1:");
                fprintf(stderr," Cell %d has %d repB1\n",c->num,c->repB1);
                goto cleanup;
        }
        c->repB1 = 0;
        c->B1 = 1;
        c->dimer += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
}

int
AutoActivateB1(CELL * c)            /*  B1+b1 -> autoB1 a_mu[23]*/
{
        int flag = 0;
        int affected = 8;
        int i;
        int todo[8] = {activateB1,repressB1,transcribeAutoB1,unComplexb1,
                        decayb1Complex,autoActivateB1,unAutoActivateB1,activateB2};

        if(c->B1 < 1 || c->b1plex < 1) {
           fprintf(stderr,"Error in AutoActivateB1:");
           fprintf(stderr," Cell %d has %d B1 and %d b1plex\n",
                        c->num,c->B1,c->b1plex);
           goto cleanup;
        }
        c->B1 = 0;
        c->b1plex -= 1;
        c->autoB1 = 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        flag = 1;
cleanup:
        return flag;
}

int
UnAutoActivateB1(CELL *c)                   /*  autoB1 -> B1, b1 a_mu[24]*/
```

```
{
        int flag = 0;
        int affected = 8;
        int i;
        int todo[8] = {activateB1,repressB1,transcribeAutoB1,unComplexb1,
                decayb1Complex,autoActivateB1,unAutoActivateB1,activateB2};

        if(c->autoB1 < 1) {
                fprintf(stderr,"Error in UnAutoActivateB1:");
                fprintf(stderr," Cell %d has %d autoB1\n",c->num,c->autoB1);
                goto cleanup;
        }
        c->B1 = 1;
        c->b1plex += 1;
        c->autoB1 = 0;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        flag = 1;
cleanup:
        return flag;
}

int
TranscribeAutoB1(CELL * c)                      /*  autoB1 -> mB1 a_mu[25]*/
{
        int flag = 0;
        int affected = 8;
        int i;
        int todo[8] = {activateB1,superActivateB1,translateB1,
                        unAutoActivateB1,repressB1,autoActivateB1,
                    transcribeAutoB1,decaymB1};
        if(c->autoB1 < 1) {
                fprintf(stderr,"Error in TranscribeAutoSuperB1:");
                fprintf(stderr," Cell %d has %d autoB1.\n", c->num,c->autoB1);
                goto cleanup;
        }
        c->mB1 += 2;            /* This should be changed to 5 or so */
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        flag = 1;

cleanup:
        return flag;
}
```

```
int
Decayb1(CELL *c)                /*  b1 -> null a_mu[26]*/
{
        int flag = 0;
        int affected = 3;
        int i;
        int todo[3] = {decayb1,complexb1,repressKrox};
        if(c->b1 < 1) {
                fprintf(stderr,"Error in Decayb1:");
                fprintf(stderr," Cell %d has %d b1.\n",c->num,c->b1);
                goto cleanup;
        }
        c->b1 -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}

int
ActivateB2(CELL *c)                     /* B2 + b1 -> actB2 a_mu[27]*/
{
        int flag = 0;
        int affected = 8;
        int r4 = (c->id == R4);
        int r5 = (c->id == R5);
        int i;
        int todo[8] = {autoActivateB1,decayb1,activateB2,unActivateB2,
          transcribeB2,decaykrox,unComplexb1,decayb1Complex};
        if(r4) {
                if(c->B2 < 1 || c->b1plex < 1) {
                        fprintf(stderr,"Error in ActivateB2:");
                        fprintf(stderr," Cell %d has %d B2 and %d b1plex\n",
                                c->num,c->B2,c->b1plex);
                        goto cleanup;
                }
                c->b1plex -= 1;
        } else if(r5) {
                if(c->B2 < 1 || c->krox < 1) {
                        fprintf(stderr,"Error in ActivateB2:");
                        fprintf(stderr," Cell %d has %d B2 and %d krox\n",
                                c->num,c->B2,c->krox);
```

```
                goto cleanup;
            }
            c->krox -= 1;
        } else {
            fprintf(stderr,"Error in ActivateB2: Called in R3\n");
            //goto cleanup;
        }

        c->B2 = 0;
        c->actB2 = 1;
        for(i = 0; i < affected; i++) {
            (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}

int
UnActivateB2(CELL *c)                   /* actB2-> B2 ,b1 a_mu[28]*/
{
        int flag = 0;
        int affected = 8;
        int r4 = (c->id == R4);
        int r5 = (c->id == R5);
        int i;
        int todo[8] = {autoActivateB1,decayb1,activateB2,unActivateB2,

transcribeB2,decaykrox,unComplexb1,decayb1Complex};

        if(c->actB2 < 1) {
            fprintf(stderr,"Error in UnActivateB2:");
            fprintf(stderr," Cell %d has %d actB2 and %d b1plex\n",
          c->num,c->actB2,c->b1plex);
            goto cleanup;
        }
        if(r4) {
            c->b1plex += 1;
        } else if (r5) {
            c->krox += 1;
        } else {
            fprintf(stderr,"Error in UnActivateB2: Called in R3\n");
        }
        c->B2 = 1;
        c->actB2 = 0;
```

```
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
TranscribeB2(CELL * c)			/*  actB2 -> mB2 a_mu[29]*/
{

	int flag = 0;
	int affected = 9;
	int i;
	int todo[9] = {activateB2,unActivateB2,transcribeB2,translateB2,

	decaymB2,decaykrox,decayb1Complex,unComplexb1,
					autoActivateB1};

	if(c->actB2 < 1) {
		fprintf(stderr,"Error in TranscribeB2:");
		fprintf(stderr," Cell %d has %d actB2.\n", c->num,c->actB2);
		goto cleanup;
	}
	c->mB2 += 1;

	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;

cleanup:
	return flag;
}

int
DecaymB2(CELL * c)			/* mA1 -> null  a_mu[30]*/
{

	int flag = 0;
	int affected = 3;
	int i;
	int todo[3] = {transcribeB2,translateB2,decaymB2};
```

```
	if(c->mB2 < 1) {
		fprintf(stderr,"Error in DecaymB2:");
		fprintf(stderr," Cell %d has %d mA1\n", c->num,c->mB2);
		goto cleanup;
	}

	c->mB2 -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
TranslateB2(CELL * c)			/*  mB2 -> b2 a_mu[31]*/
{
	int flag = 0;
	int affected = 3;
	int i;
	int todo[3] = {translateB2,decayb2,decaymB2};
	if(c->mB2 < 1) {
		fprintf(stderr,"Error in TranslateB2:");
		fprintf(stderr," Cell %d has %d mB2\n", c->num,c->mB2);
		goto cleanup;
	}
	c->b2 += 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;

cleanup:
	return flag;
}

int
Decayb2(CELL *c)		/*  b2 -> null a_mu[32]*/
{
	int flag = 0;
	int affected = 1;
	int i;
```

```
	int todo[1] = {decayb2};
	if(c->b2 < 1) {
		fprintf(stderr,"Error in Decayb2:");
		fprintf(stderr," Cell %d has %d b2.\n",c->num,c->b2);
		goto cleanup;
	}
	c->b2 -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

/* a_mu[33] is Divide */

int
ActivateKrox(CELL *c)		/* a1 + Krox -> actKrox a_mu[34] */
{
	int flag = 0;
	int affected = 4;
	int i;
	int todo[4] = {activateKrox,unActivateKrox,transcribeKrox,repressKrox};
	if(c->Krox < 1) {
		fprintf(stderr,"Error in ActivateKrox:");
		fprintf(stderr," Cell %d has %d Krox.\n",
			c->num,c->Krox);
		goto cleanup;
	}
	c->Krox = 0;
	c->actKrox = 1;
	for(i = 0; i <  affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
UnActivateKrox(CELL *c)
{
	int flag = 0;
```

```
	int affected = 4;
	int i;
	int todo[4] = {activateKrox,unActivateKrox,transcribeKrox,repressKrox};
	if(c->actKrox < 1) {
		fprintf(stderr,"Error in UnActivateKrox:");
		fprintf(stderr," Cell %d has %d Krox\n", c->num,c->Krox);
		goto cleanup;
	}
	c->Krox = 1;
	c->actKrox = 0;
	for(i = 0; i <  affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
TranscribeKrox(CELL *c)
{
	int flag = 0;
	int affected = 6;
	int todo[6] = {activateKrox,unActivateKrox,transcribeKrox,repressKrox,
					translateKrox,decaymKrox};
	int i;

	if(c->actKrox < 1) {
		fprintf(stderr,"Error in TranscribeKrox:");
		fprintf(stderr," Cell %d has %d actKrox\n", c->num,c->actKrox);
		goto cleanup;
	}
	c->mKrox += 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
	cleanup:
	return flag;
}

int
DecaymKrox(CELL *c)
{
```

```
	int flag = 0;
	int affected = 3;
	int todo[3] = {transcribeKrox,translateKrox,decaymKrox};
	int i;

	if(c->mKrox < 1) {
		fprintf(stderr,"Error in DecaymKrox:");
		fprintf(stderr," Cell %d has %d mKrox\n", c->num,c->mKrox);
		goto cleanup;
	}

	c->mKrox -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
RepressKrox(CELL *c)
{
	int flag = 0;
	int affected = 10;
	int todo[10] = {decaya1,superActivateB1,activateKrox,unRepressKrox,

	unActivateKrox,transcribeKrox,repressKrox,complexb1,
					complexa1,decayb1};
	int i;

	if(c->Krox < 1 || (c->b1 < 1 && c->a1 < 1)) {
		fprintf(stderr,"Error in RepressKrox:");
		fprintf(stderr," Cell %d (%d) has %d Krox,%d a1 and %d b1\n",
					c->num,c->id+3,c->Krox,c->a1,c->b1);
		goto cleanup;
	}
	c->repKrox = 1;
	c->Krox = 0;

	if(c->a1 > c->b1) {
		c->a1 -= 1;
	   c->krep = A;
	} else {
		c->b1 -= 1;
```

```
            c->krep = B;
        }

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
}

int
UnRepressKrox(CELL *c)
{
        int flag = 0;
        int affected = 10;
        int i;
        int todo[10] = {decaya1,superActivateB1,activateKrox,unRepressKrox,

        unActivateKrox,transcribeKrox,repressKrox,complexb1,
                                        decayb1,complexa1};

        if(c->repKrox < 1) {
                fprintf(stderr,"Error in UnRepressKrox:");
                fprintf(stderr," Cell %d has %d Krox and %d a1\n",
                                                c->num,c->Krox,c->a1);
                goto cleanup;
        }
        c->repKrox = 0;
        c->Krox = 1;
        if(c->krep == A) {
                c->a1 += 1;
        } else {
                c->b1 += 1;
        }

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;

cleanup:
        return flag;
```

```
}

int
TranslateKrox(CELL *c)
{
	int flag = 0;
	int affected = 4;
	int i;
	int todo[4] = {translateKrox,decaykrox,decaymKrox,activateB2};
	if(c->mKrox < 1) {
		fprintf(stderr,"Error in TranslateKrox:");
		fprintf(stderr," Cell %d has %d mKrox\n", c->num,c->mKrox);
		goto cleanup;
	}
	c->krox += 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;

cleanup:
	return flag;
}

int
Decaykrox(CELL *c)
{
	int flag = 0;
	int affected = 2;
	int i;
	int todo[2] = {decaykrox,activateB2};
	if(c->krox < 1) {
		fprintf(stderr,"Error in DecayKrox:");
		fprintf(stderr," Cell %d has %d krox.\n",c->num,c->krox);
		goto cleanup;
	}
	c->krox -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}
```

```
int
BindRXR(CELL *c)      /* a_mu[42] */
{
        int flag = 0;
        int affected = 7;
        int i;
        int todo[7]= {bindRAR,decayRA,decayRXR,bindRXR,unbindBRXR,
                                decayBRXR,dimerize};
        if(c->ra < 1 || c->rxr < 1) {
                fprintf(stderr,"Error in BindRXR:");
                fprintf(stderr," Cell %d has %d ra and %d rxr\n",c->num,c->ra,c->rxr);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->ra -= 1;
        c->rxr -= 1;
        c->brxr += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        c->d_ra = D_ra*(c->ra);
        flag = 1;
cleanup:
        return flag;

}

int
MakeRXR(CELL *c)  /* a_mu[43] */
{
        int affected = 2;
        int todo[2] = {bindRXR,decayRXR};
        int flag = 0;
        int i;
        c->rxr += 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        flag = 1;
        return flag;
}
```

```
int
DecayRXR(CELL *c)     /* a_mu[44] */
{
        int affected = 2;
        int todo[2] = {bindRXR,decayRXR};
        int flag = 0;
        int i;
        if(c->rxr < 1) {
                fprintf(stderr,"Error in DecayRXR:");
                fprintf(stderr," Cell %d has %d RXR\n", c->num,c->rxr);
                goto cleanup;
        }
        c->rxr -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }
        flag = 1;

cleanup:
        return flag;

}

int
UnbindBRXR(CELL *c)   /* a_mu[45] */
{
        int flag = 0;
        int affected = 7;
        int i;
        int todo[7]= {bindRAR,decayRA,decayRXR,bindRXR,unbindBRXR,
                                decayBRXR,dimerize};
        if(c->brxr < 1) {
                fprintf(stderr,"Error in UnbindBRXR:");
                fprintf(stderr," Cell %d has %d brxr.\n",c->num,c->brxr);
                goto cleanup;
        }
/* Change the relevant quantities */
        c->ra += 1;
        c->rxr += 1;
        c->brxr -= 1;
        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        c->d_ra = D_ra*(c->ra);
```

```
	flag = 1;
cleanup:
	return flag;

}

int
DecayBRXR(CELL *c)    /* a_mu[46] */
{
	int affected = 3;
	int todo[3] = {unbindBRXR,decayBRXR,dimerize};
	int flag = 0;
	int i;
	if(c->brxr < 1) {
		fprintf(stderr,"Error in DecayBRXR:");
		fprintf(stderr," Cell %d has %d brxr.\n",c->num,c->brxr);
		goto cleanup;
	}
/* Change the relevant quantities */
	c->brxr -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
	cleanup:

	return flag;

}

int
Dimerize(CELL *c)     /* a_mu[47] */
{
	int flag = 0;
	int affected = 11;
	int i;
	int todo[11]= {unbindBRAR,decayBRAR,unbindBRXR,decayBRXR,
			activateA1,activateB1,repressB1,decayDimer,
			dimerize,unDimerize,transcribeA1};

	if(c->brar < 1 || c->brxr < 1) {
		fprintf(stderr,"Error in Dimerize:");
		fprintf(stderr," Cell %d has %d brar and %d brxr\n",
			c->num,c->brar,c->brxr);
```

```
                goto cleanup;
        }

/* Change the relevant quantities */
        c->brar -= 1;
        c->brxr -= 1;
        c->dimer += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}

int
UnDimerize(CELL *c)   /* a_mu[48] */
{
        int flag = 0;
        int affected = 11;
        int i;
        int todo[11]= {unbindBRAR,decayBRAR,unbindBRXR,decayBRXR,dimerize,
                        activateA1,activateB1,repressB1,decayDimer,
                        unDimerize,transcribeA1};


        if(c->dimer < 1) {
                fprintf(stderr,"Error in UnDimerize:");
                fprintf(stderr," Cell %d has %d dimers\n", c->num,c->dimer);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->brar += 1;
        c->brxr += 1;
        c->dimer -= 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
```

```
}

int
DecayDimer(CELL *c)    /* a_mu[49] */
{
        int flag = 0;
        int affected = 6;
        int todo[6]= {activateA1,activateB1,repressB1,decayDimer,unDimerize,
                                transcribeA1};
        int i;

        if(c->dimer < 1) {
                fprintf(stderr,"Error in DecayDimer:");
                fprintf(stderr," Cell %d has %d dimers\n", c->num,c->dimer);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->dimer -= 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;

}

int
Complexa1(CELL *c)      /* a_mu[50] */
{
        int flag = 0;
        int affected = 8;
        int i;
        int todo[8]= {complexb1,superActivateB1,decaya1Complex,
                                decayComplex,decaya1,complexa1,repressKrox,
                                unComplexa1};

        if(c->a1 < 1 || c->plex < 1) {
                fprintf(stderr,"Error in Complexa1:");
                fprintf(stderr," Cell %d has %d a1 and %d complexes\n"
                        ,c->num,c->a1,c->plex);
                goto cleanup;
```

```
        }

/* Change the relevant quantities */
        c->a1 -= 1;
        c->plex -= 1;
        c->a1plex += 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}

int
Uncomplexa1(CELL *c)    /* a_mu[51] */
{
        int flag = 0;
        int affected = 8;
        int i;
        int todo[8]= {complexb1,superActivateB1,decaya1Complex,
                                decayComplex,decaya1,complexa1,repressKrox,
                                unComplexa1};

        if(c->a1plex < 1) {
                fprintf(stderr,"Error in UnComplexa1:");
                fprintf(stderr," Cell %d has %d a1plex\n" ,c->num,c->a1plex);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->a1 += 1;
        c->plex += 1;
        c->a1plex -= 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}
```

```
int
Decaya1Complex(CELL *c) /* a_mu[52] */
{
        int flag = 0;
        int affected = 3;
        int i;
        int todo[3]= {superActivateB1,decaya1Complex,unComplexa1};

        if(c->a1plex < 1) {
                fprintf(stderr,"Error in Decaya1Complex:");
                fprintf(stderr," Cell %d has %d a1plex\n" ,c->num,c->a1plex);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->a1plex -= 1;

        for(i = 0; i < affected; i++) {
                (Update)[todo[i]](c);
        }

        flag = 1;
cleanup:
        return flag;
}

int
Complexb1(CELL *c)      /* a_mu[53] */
{
        int flag = 0;
        int affected = 9;
        int i;
        int todo[9]= {complexb1,autoActivateB1,activateB2,complexa1,
                                unComplexb1,decayb1Complex,decayb1,decayComplex,
                                repressKrox};

        if(c->b1 < 1 || c->plex < 1) {
                fprintf(stderr,"Error in Complexb1:");
                fprintf(stderr," Cell %d has %d b1 and %d complexes\n"
                        ,c->num,c->b1,c->plex);
                goto cleanup;
        }

/* Change the relevant quantities */
        c->b1 -= 1;
        c->plex -= 1;
```

```
	c->b1plex += 1;

	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
Uncomplexb1(CELL *c)	/* a_mu[54] */
{
	int flag = 0;
	int affected = 9;
	int i;
	int todo[9]= {complexb1,autoActivateB1,activateB2,complexa1,
				unComplexb1,decayb1Complex,decayb1,decayComplex,
				repressKrox};

	if(c->b1plex < 1) {
		fprintf(stderr,"Error in UnComplexb1:");
		fprintf(stderr," Cell %d has %d b1\n",c->num,c->b1plex);
		goto cleanup;
	}

/* Change the relevant quantities */
	c->b1 += 1;
	c->plex += 1;
	c->b1plex -= 1;

	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
Decayb1Complex(CELL *c)  /* a_mu[55] */
{
	int flag = 0;
	int affected = 4;
```

```
	int i;
	int todo[4]= {autoActivateB1,activateB2,unComplexb1,decayb1Complex};

	if(c->b1plex < 1) {
		fprintf(stderr,"Error in Decayb1Complex:");
		fprintf(stderr," Cell %d has %d b1\n",c->num,c->b1plex);
		goto cleanup;
	}

/* Change the relevant quantities */
	c->b1plex -= 1;

	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}

	flag = 1;
cleanup:
	return flag;
}

int
MakeComplex(CELL *c)
{
	int affected = 3;
	int todo[3] = {complexa1,complexb1,decayComplex};
	int flag = 0;
	int i;
	c->plex += 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
	}
	flag = 1;
	return flag;
}

int
DecayComplex(CELL *c)
{
	int affected = 3;
	int todo[3] = {complexa1,complexb1,decayComplex};
	int flag = 0;
	int i;
	c->plex -= 1;
	for(i = 0; i < affected; i++) {
		(Update)[todo[i]](c);
```

```
        }
        flag = 1;
        return flag;

}


int
RA_Diffusion(CELL * c)   /*  RA diffusion; Source located at xi = 0 */
{
        int ra;
        int flag = 0;

        if((c->next != (CELL*) NULL) && (c->ra > 0)) {
                ra = ((c+1)->ra += 1);
                (c+1)->d_ra = D_ra*ra;
                (c+1)->a_mu[bindRAR] = C_mu[bindRAR]*ra*(c+1)->rar;
                (c+1)->a_mu[bindRXR] = C_mu[bindRXR]*ra*(c+1)->rxr;
                (c+1)->a_mu[decayRA] = C_mu[decayRA]*ra;
                ra = (c->ra -= 1);
                c->d_ra = D_ra*ra;
                c->a_mu[bindRAR] = C_mu[bindRAR]*ra*c->rar;
                c->a_mu[bindRXR] = C_mu[bindRXR]*ra*c->rxr;
                c->a_mu[decayRA] = C_mu[decayRA]*ra;
                flag = 1;
        }
        else {
                printf("Can't diffuse in cell %d, have %d ra.\n",c->num,c->ra);
        }
        return flag;
}
```

UpdateAmu.h

```
#include "Hox.h"

#ifndef __UPDATEAMU_H
#define __UPDATEAMU_H

void UpdateMakeRA(CELL *);
void UpdateBindRAR(CELL *);
void UpdateDecayRA(CELL *);
void UpdateMakeRAR(CELL *);
void UpdateDecayRAR(CELL *);
void UpdateUnBindBRAR(CELL *);
```

```
void UpdateDecayBRAR(CELL *);
void UpdateActivateA1(CELL *);
void UpdateUnActivateA1(CELL *);
void UpdateTranscribeA1(CELL *);
void UpdateDecaymA1(CELL *);
void UpdateTranslateA1(CELL *);
void UpdateDecaya1(CELL *);
void UpdateActivateB1(CELL *);
void UpdateUnActivateB1(CELL *);
void UpdateTranscribeB1(CELL *);
void UpdateDecaymB1(CELL *);
void UpdateTranslateB1(CELL *);
void UpdateSuperActivateB1(CELL *);
void UpdateUnSuperActivateB1(CELL *);
void UpdateTranscribeSuperB1(CELL *);
void UpdateRepressB1(CELL *);
void UpdateUnRepressB1(CELL *);
void UpdateAutoActivateB1(CELL *);
void UpdateUnAutoActivateB1(CELL *);
void UpdateTranscribeAutoB1(CELL *);
void UpdateDecayb1(CELL *);
void UpdateActivateB2(CELL *);
void UpdateUnActivatedB2(CELL *);
void UpdateTranscribeB2(CELL *);
void UpdateDecaymB2(CELL *);
void UpdateTranslateB2(CELL *);
void UpdateDecayb2(CELL *);
void UpdateDivide(CELL *);
void UpdateActivateKrox(CELL *);
void UpdateUnActivateKrox(CELL *);
void UpdateTranscribeKrox(CELL *);
void UpdateDecaymKrox(CELL *);
void UpdateRepressKrox(CELL *);
void UpdateUnRepressKrox(CELL *);
void UpdateTranslateKrox(CELL *);
void UpdateDecaykrox(CELL *);
void UpdateBindRXR(CELL *);
void UpdateMakeRXR(CELL *);
void UpdateDecayRXR(CELL *);
void UpdateUnbindBRXR(CELL *);
void UpdateDecayBRXR(CELL *);
void UpdateDimerize(CELL *);
void UpdateUnDimerize(CELL *);
void UpdateDecayDimer(CELL *);
void UpdateComplexa1(CELL *);
void UpdateUnComplexa1(CELL *);
```

```
void UpdateDecaya1Complex(CELL *);
void UpdateComplexb1(CELL *);
void UpdateUnComplexb1(CELL *);
void UpdateDecayb1Complex(CELL *);
void UpdateMakeComplex(CELL *);
void UpdateDecayComplex(CELL *);

typedef  void (*UpdateAmu)();

static UpdateAmu       Update[NUM_FUNCS] = {
UpdateMakeRA,
UpdateBindRAR,
UpdateDecayRA,
UpdateMakeRAR,
UpdateDecayRAR,
UpdateUnBindBRAR,
UpdateDecayBRAR,
UpdateActivateA1,
UpdateUnActivateA1,
UpdateTranscribeA1,
UpdateDecaymA1,
UpdateTranslateA1,
UpdateDecaya1,
UpdateActivateB1,
UpdateUnActivateB1,
UpdateTranscribeB1,
UpdateDecaymB1,
UpdateTranslateB1,
UpdateSuperActivateB1,
UpdateUnSuperActivateB1,
UpdateTranscribeSuperB1,
UpdateRepressB1,
UpdateUnRepressB1,
UpdateAutoActivateB1,
UpdateUnAutoActivateB1,
UpdateTranscribeAutoB1,
UpdateDecayb1,
UpdateActivateB2,
UpdateUnActivatedB2,
UpdateTranscribeB2,
UpdateDecaymB2,
UpdateTranslateB2,
UpdateDecayb2,
UpdateDivide,
UpdateActivateKrox,
UpdateUnActivateKrox,
```

```
UpdateTranscribeKrox,
UpdateDecaymKrox,
UpdateRepressKrox,
UpdateUnRepressKrox,
UpdateTranslateKrox,
UpdateDecaykrox,
UpdateBindRXR,
UpdateMakeRXR,
UpdateDecayRXR,
UpdateUnbindBRXR,
UpdateDecayBRXR,
UpdateDimerize,
UpdateUnDimerize,
UpdateDecayDimer,
UpdateComplexa1,
UpdateUnComplexa1,
UpdateDecaya1Complex,
UpdateComplexb1,
UpdateUnComplexb1,
UpdateDecayb1Complex,
UpdateMakeComplex,
UpdateDecayComplex};

#endif
```

UpdateAmu.c

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include "Hox.h"
#include "header.h"
#include "UpdateAmu.h"


void                                    /* 0 */
UpdateMakeRA(CELL *c)
{

}

void                                        /* 1 */
UpdateBindRAR(CELL *c)
{
```

```
        c->a_mu[bindRAR] = C_mu[bindRAR]*(c->ra)*(c->rar);
}

void                                    /* 2 */
UpdateDecayRA(CELL *c)
{
  c->a_mu[decayRA]   = C_mu[decayRA]*(c->ra);
}

void                                    /* 3 */
UpdateMakeRAR(CELL *c)
{

}

void
UpdateDecayRAR(CELL *c)         /* 4 */
{
   c->a_mu[decayRAR]  = C_mu[decayRAR]*(c->rar);
}

void
UpdateUnBindBRAR(CELL *c)       /* 5 */
{
        c->a_mu[unbindBRAR]  = C_mu[unbindBRAR]*(c->brar);
}

void
UpdateDecayBRAR(CELL *c)                /* 6 */
{
        c->a_mu[decayBRAR]  = C_mu[decayBRAR]*(c->brar);
}

void
UpdateActivateA1(CELL *c) /* 7 */
{
//    int dimer = c->dimer;
//    c->a_mu[activateA1]  = C_mu[activateA1]*pow((double)dimer,a1hill)/
//                        (K[1]+pow((double)dimer,a1hill))*(dimer)*(c->A1);
}

void
UpdateUnActivateA1(CELL *c)       /* 8 */
{
//    c->a_mu[unActivateA1] = C_mu[unActivateA1]*(c->actA1);
}
```

```
void
UpdateTranscribeA1(CELL *c) /* 9 */
{
//        c->a_mu[transcribeA1] = C_mu[transcribeA1]*(c->actA1);
/* This version is for the updated model */
          c->a_mu[transcribeA1] = C_mu[transcribeA1]*(c->dimer);
}

void
UpdateDecaymA1(CELL *c) /* 10 */
{
          c->a_mu[decaymA1] = C_mu[decaymA1]*(c->mA1);
}

void
UpdateTranslateA1(CELL *c)          /* 11 */
{
          c->a_mu[translateA1] = C_mu[translateA1]*(c->mA1);
}

void
UpdateDecaya1(CELL *c)                        /* 12 */
{
   c->a_mu[decaya1]  = C_mu[decaya1]*(c->a1);
}

void
UpdateActivateB1(CELL *c)           /* 13 */
{
   int dimer = c->dimer;
   c->a_mu[activateB1]  = C_mu[activateB1]*pow((double)dimer,b1hill)/
                     (K[2]+pow((double)dimer,b1hill))*dimer*c->B1;
}

void
UpdateUnActivateB1(CELL *c)                   /* 15 */
{
   c->a_mu[unActivateB1] = C_mu[unActivateB1]*(c->actB1);
}

void
UpdateTranscribeB1(CELL *c)                   /* 15 */
{
   c->a_mu[transcribeB1] = C_mu[transcribeB1]*(c->actB1);
```

```
}

void
UpdateDecaymB1(CELL *c)                    /* 16 */
{
        c->a_mu[decaymB1] = C_mu[decaymB1]*(c->mB1);
}

void
UpdateTranslateB1(CELL *c)                 /* 17 */
{
   c->a_mu[translateB1] = C_mu[translateB1]*(c->mB1);
}

void
UpdateSuperActivateB1(CELL *c)             /* 18 */
{
   int a1plex = c->a1plex;
   int id = (c->id == R4);
   c->a_mu[superActivateB1]  = id*C_mu[superActivateB1]*
                        pow((double)a1plex,a1hill)/
                        (K[3]+pow((double)a1plex,a1hill))* a1plex*(c->actB1);
}

void
UpdateUnSuperActivateB1(CELL *c)           /* 19 */
{
   c->a_mu[unSuperActivateB1] = C_mu[unSuperActivateB1]*(c->superactB1);
}

void
UpdateTranscribeSuperB1(CELL *c)           /* 20 */
{
   c->a_mu[transcribeSuperB1] = C_mu[transcribeSuperB1]*(c->superactB1);
}

void
UpdateRepressB1(CELL *c)             /* 21 */
{
   int dimer = c->dimer;
   c->a_mu[repressB1]  = C_mu[repressB1]*c->B1*dimer/
                     (K[6]+pow((double)dimer,rephill));
}

void
UpdateUnRepressB1(CELL *c)                 /* 22 */
```

```
{
   c->a_mu[unRepressB1] = C_mu[unRepressB1]*(c->repB1);
}

void
UpdateAutoActivateB1(CELL *c)              /* 23 */
{
   int b1plex = c->b1plex;
   int id = (c->id == R4);
   c->a_mu[autoActivateB1]  = id*C_mu[autoActivateB1]*
                                        pow((double)b1plex,b1auto)/
                                        (K[4]+pow((double)b1plex,b1auto))*
                                        b1plex*(c->B1);
}

void
UpdateUnAutoActivateB1(CELL *c)          /* 24 */
{
   c->a_mu[unAutoActivateB1] = C_mu[unAutoActivateB1]*(c->autoB1);
}

void
UpdateTranscribeAutoB1(CELL *c) /* 25 */
{
   c->a_mu[transcribeAutoB1] = C_mu[transcribeAutoB1]*(c->autoB1);
}
void
UpdateDecayb1(CELL *c)                     /* 26 */
{
   c->a_mu[decayb1]  = C_mu[decayb1]*(c->b1);
}

void
UpdateActivateB2(CELL *c)          /* 27 */
{
   int act;
   int r3 = (c->id == R3);
   int r4 = (c->id == R4);
   int r5 = (c->id == R5);
         if(r4) act = c->b1plex;
         else if(r5) act = c->krox;
         else act = c->krox;
   c->a_mu[activateB2]  = !r3*C_mu[activateB2]*pow((double)act,b2hill)/
                     (K[5]+pow((double)act,b2hill))*act*(c->B2);
}
```

```
void
UpdateUnActivatedB2(CELL *c)                /* 28 */
{
   c->a_mu[unActivateB2] = C_mu[unActivateB2]*(c->actB2);
}

void
UpdateTranscribeB2(CELL *c)                 /* 29 */
{
   c->a_mu[transcribeB2] = C_mu[transcribeB2]*(c->actB2);
}

void
UpdateDecaymB2(CELL *c)                     /* 30 */
{
        c->a_mu[decaymB2] = C_mu[decaymB2]*(c->mB2);
}

void
UpdateTranslateB2(CELL *c)                  /* 31 */
{
   c->a_mu[translateB2] = C_mu[translateB2]*(c->mB2);
}

void
UpdateDecayb2(CELL *c)                      /* 32 */
{
   c->a_mu[decayb2]  = C_mu[decayb2]*(c->b2);
}

void
UpdateDivide(CELL *c)                       /* 33 */
{
        /* no changes needed */
}

void
UpdateActivateKrox(CELL *c)                         /* 34 */
{
   int r3 = (c->id == R3);
   c->a_mu[activateKrox]  = !r3*C_mu[activateKrox]*(c->Krox);
}

void
UpdateUnActivateKrox(CELL *c) /* a_mu[35] */
{
```

```
   c->a_mu[unActivateKrox] = C_mu[unActivateKrox]*(c->actKrox);
}

void
UpdateTranscribeKrox(CELL *c) /* a_mu[36] */
{
   c->a_mu[transcribeKrox] = C_mu[transcribeKrox]*(c->actKrox);
}

void
UpdateDecaymKrox(CELL *c)     /* a_mu[37] */
{
        c->a_mu[decaymKrox] = C_mu[decaymKrox]*(c->mKrox);
}

void
UpdateRepressKrox(CELL *c)    /* a_mu[38] */
{

   int b1 = c->b1;
   int a1 = c->a1;
   int max = (a1 > b1) ? a1 : b1;

   c->a_mu[repressKrox]  = C_mu[repressKrox]*c->Krox*(max)/
                  (K[6]+pow((double)(max),rephill));

}

void
UpdateUnRepressKrox(CELL *c)  /* a_mu[39] */
{
   int r3 = (c->id == R3);
   c->a_mu[unRepressKrox] = !r3*C_mu[unRepressKrox]*(c->repKrox);
}

void
UpdateTranslateKrox(CELL *c)  /* a_mu[40] */
{
   c->a_mu[translateKrox] = C_mu[translateKrox]*(c->mKrox);
}

void
UpdateDecaykrox(CELL *c)      /* a_mu[41] */
{
   c->a_mu[decaykrox]  = C_mu[decaykrox]*(c->krox);
}
```

```
void
UpdateBindRXR(CELL *c)              /* a_mu[42] */
{
        c->a_mu[bindRXR] = C_mu[bindRXR]*(c->ra)*(c->rxr);
}

void
UpdateMakeRXR(CELL *c)  /* a_mu[43] */
{

}

void
UpdateDecayRXR(CELL *c)             /* a_mu[44] */
{
   c->a_mu[decayRXR]  = C_mu[decayRXR]*(c->rxr);
}

void
UpdateUnbindBRXR(CELL *c)       /* a_mu[45] */
{
        c->a_mu[unbindBRXR]  = C_mu[unbindBRXR]*(c->brxr);
}

void
UpdateDecayBRXR(CELL *c)        /* a_mu[46] */
{
        c->a_mu[decayBRXR]  = C_mu[decayBRXR]*(c->brxr);
}

void
UpdateDimerize(CELL *c)             /* a_mu[47] */
{
        c->a_mu[dimerize] = C_mu[dimerize]*(c->brar)*(c->brxr);

}

void
UpdateUnDimerize(CELL *c)   /* a_mu[48] */
{
        c->a_mu[unDimerize] = C_mu[unDimerize]*(c->dimer);
}

void
UpdateDecayDimer(CELL *c)       /* a_mu[49] */
```

```
{
	c->a_mu[decayDimer] = C_mu[decayDimer]*(c->dimer);
}

void
UpdateComplexa1(CELL *c)
{
	c->a_mu[complexa1] = C_mu[complexa1]*(c->a1)*(c->plex);
}

void
UpdateUnComplexa1(CELL *c)
{
	c->a_mu[unComplexa1] = C_mu[unComplexa1]*(c->a1plex);
}

void
UpdateDecaya1Complex(CELL *c)
{
	c->a_mu[decaya1Complex] = C_mu[decaya1Complex]*(c->a1plex);
}

void
UpdateComplexb1(CELL *c)
{
	c->a_mu[complexb1] = C_mu[complexb1]*(c->b1)*(c->plex);
}

void
UpdateUnComplexb1(CELL *c)
{
	c->a_mu[unComplexb1] = C_mu[unComplexb1]*(c->b1plex);
}

void
UpdateDecayb1Complex(CELL *c)
{
	c->a_mu[decayb1Complex] = C_mu[decayb1Complex]*(c->b1plex);
}

void
UpdateMakeComplex(CELL *c)
{

}
```

```
void
UpdateDecayComplex(CELL *c)
{
        c->a_mu[decayComplex] = C_mu[decayComplex]*(c->plex);
}
```

ranlib

The ranlib routines used in this program are in the public domain and can be found at http://www.netlib.org/random/ and are fully described in the literature (L'Ecuyer et al., 1991).

Makefile

```
CC = gcc

CFLAGS = -O2 -Wall
TARGET = a.out

LIBS = -lm



SRCS = Hox.c main.c write_gnu_data_file.c inputs.c UpdateAmu.c ll.c ranlib.c com.c
OBJS = Hox.o main.o write_gnu_data_file.o inputs.o UpdateAmu.o ll.o ranlib.o com.o

$(TARGET):  $(OBJS)
            $(CC) -o $(TARGET) $(CFLAGS) $(LFLAGS) $(OBJS) $(LIBS)
            chmod 755 $(TARGET)

main.o:     main.c Hox.h header.h
            $(CC) -c $(CFLAGS) $(LFLAGS) main.c

Hox.o:      Hox.c Hox.h header.h
            $(CC) -c $(CFLAGS) $(LFLAGS) Hox.c

UpdateAmu.o:UpdateAmu.c UpdateAmu.h header.h
            $(CC) -c $(CFLAGS) $(LFLAGS) UpdateAmu.c

ranlib.o:   ranlib.c ranlib.h
            $(CC) -c $(CFLAGS) $(LFLAGS) ranlib.c

com.o:      com.c ranlib.h
```

```
		$(CC) -c $(CFLAGS) $(LFLAGS) com.c

inputs.o:	inputs.c Hox.h header.h
		$(CC) -c $(CFLAGS) $(LFLAGS) inputs.c

ll.o:		ll.c Hox.h header.h
		$(CC) -c $(CFLAGS) $(LFLAGS) ll.c

write_gnu_data_file.o:		write_gnu_data_file.c
		$(CC) -c $(CFLAGS) $(LFLAGS) write_gnu_data_file.c

clean:
		rm -f $(TARGET) $(OBJS) count
```

Appendix D: Mathematica Source Code

The *Mathematica* package that was used for making the movies is included below. The notebook can be found on the CD-ROM as well.

Basic Enzyme Reaction

The following source code was used to generate the data used for Figures 2.1 and 2.2.

```
MM[inputsub_, inputenz_, end_,k_List] :=
	Module[{kf = k[[1]], kb = k[[2]], k2 = k[[3]], enz = inputenz,
			sub = inputsub, com = 0, pro = 0, t = 0.0, tt = {},  dat = {}, s},
		While[t < end && (sub > 0 || com > 0),
			amu = {kf*sub*enz, kb*com, k2*com};
			a0 = Plus @@ amu;
			r1 = Random[ ];
			t += - Log[r1]/a0;
			r2 = Random[ ];
			picker = r2*a0;
			s = Drop[FoldList[Plus, 0, amu], 1];
			Which[picker < s[[1]],
					sub -= 1; enz -= 1; com += 1,
				picker < s[[2]],
					sub += 1; enz += 1; com -= 1,
				picker < s[[3]],
```

```
                                    enz += 1; com -= 1; pro += 1
                ];
                AppendTo[tt, t];
                AppendTo[dat, {sub, enz, com, pro}]
            ];
            {tt, dat}
        ]
```

Data Display Routines

Response Curves

For displaying how the levels in a particular cell of a certain specie changes over time.

```
Needs["Graphics`MultipleListPlot`"];

Response[files:{___String},rhom_] := Module[{data = {},name,i,j,m,l},
   clrs = {RGBColor[1,0,0],RGBColor[0,1,0],
      RGBColor[0,0,1],RGBColor[0,1,1],RGBColor[1,0,1],RGBColor[0,0,0],
      RGBColor[1,.5,0],
      RGBColor[0,.5,.5],RGBColor[.5,1,.5],RGBColor[1,.5,.5],
      RGBColor[.5,.5,.5]
      };
   l = Length[files];
   For[j = 1, j ≤ l, j++,
    d = ReadList[files[[j]],Real,RecordLists->True];
    If[rhom == 5,
        d = Map[Part[#,2]&,d],
            d = Map[Part[#,4]&,d]
     ];
    AppendTo[data,d];
     ];
   m = Max[data];
   For[i=1, i<=l, i++,mx = Length[data[[1,i]]];
     a =
      Table[Text[
        StyleForm[files[[i]],FontColor->clrs[[i]]],{8,12-2*i}],{i,1,
        l}]
    ];
   MultipleListPlot[Apply[Sequence,data], SymbolStyle->clrs ,
    Prolog->a]
  ]
```

Excess Variance

The routine that generates the Figures 3.10 and the like.

```
Needs["Graphics`MultipleListPlot`"];

ExcessVar[files:{___String}] := Module[{data = {},name,i,j,m,l},
   clrs = {RGBColor[1,0,0],RGBColor[0,1,0],
      RGBColor[0,0,1],RGBColor[0,1,1],RGBColor[1,0,1],RGBColor[0,0,0],
      RGBColor[1,.5,0],
      RGBColor[0,.5,.5],RGBColor[.5,1,.5],RGBColor[1,.5,.5],
      RGBColor[.5,.5,.5]
      };
   l = Length[files];
   For[j = 1, j ≤ l, j++,
    d = ReadList[files[[j]],Real,RecordLists->True];
        d = Map[Part[#,2]&,d];
    AppendTo[data,d];
    ];
   m = Max[data];
   For[i=1, i<=l, i++,mx = Length[data[[1,i]]];
     a =
      Table[Text[
         StyleForm[files[[i]],FontColor->clrs[[i]]],{8,12-2*i}],{i,1,
         l}]
   ];
   MultipleListPlot[Apply[Sequence,data], SymbolStyle->clrs , Prolog->a]
   ]
```

Level Initilaztion

The initial incarnation of the data display routines, these are still in use by J. Solomon

(personal communication).

```
Needs["Graphics`MultipleListPlot`"];

Movie[dir_,files:{___String}, num_Integer,opts___Rule] := Movie[dir,files,0,num,opts];

Options[Movie] = {Step->1};

Movie[dir_,files:{___String}, start_Integer,num_Integer,opts___Rule] :=
  Module[{data = {},name,numbers={},i,j,m,l},
    clrs = {RGBColor[1,0,0],RGBColor[0,1,0],
       RGBColor[0,0,1],RGBColor[0,1,1],RGBColor[1,0,1],RGBColor[0,0,0],
```

```
    RGBColor[1,.5,0],
    RGBColor[0,.5,.5],RGBColor[.5,1,.5],RGBColor[1,.5,.5],
    RGBColor[.5,.5,.5]
    };
  mystep=Step/.{opts}/.Options[Movie];
  l = Length[files];
  For[j = 1, j ≤ l, j++,
   data = {};
     For[i = start, i ≤ num, i+=mystep,
    name =dir<>files[[j]]<>"."<>ToString[i]<>".dat";
    d = ReadList[name,Real,RecordLists->True];
    d = Map[Last,d];
    AppendTo[data,d];
     ];
   AppendTo[numbers,data];
   ];
  m = Max[numbers];
  Which[m < 2500, scale = 100,
            m < 6000, scale = 300,
           m < 10000, scale = 500,
           True, scale = 1000];
  For[i=1, i≤(num-start)/mystep, i++,mx = Length[numbers[[1,i]]];
    a =
    Table[Text[
       StyleForm[files[[i]],FontColor->clrs[[i]]],{mx-2,
         m-i* scale}],{i,1,l}];
   MultipleListPlot[Map[Part[#,i]&,numbers],PlotRange->{-200,m},
    SymbolStyle->clrs ,
    Prolog->a]]
  ]
```

Stain Initilaztion

These routines produce the virtual dynamic *in situ* as in Figure 3.5.

```
Stain[direc_,files:{___String}, num_Integer,opts___Rule] :=
Stain[direc,files,0,num,opts];

Options[Stain] = {Step->1,FrameScale->False,Tiffs->False};

TimeSlice[data_,time_] := Module[{},
  Map[#[[time]]&,data]
];
```

```
SpecieSlice[data_,specie_] := Module[{},
   Map[#[[specie]]&,data]
];

MakeBar[specie_,num_,name_,gmx_,border_,ndir_] :=
 Module[{bar={},i,mol,scale,color,tc,x,y,gmaxx},
   mx = 1+Max[specie];
   gmaxx = gmx  + 1;
   xoff = 6;
   s = specie/(mx+1);
   s = specie;
   numdirs = Length[specie];
   Which[num == 5,
    tc = CMYKColor[0,1,1,0],
    num== 4,
    tc = CMYKColor[1,0,1,0],
    num == 3,
    tc = CMYKColor[1,1,0,0],
    num == 2,
    tc = CMYKColor[0,1,0,0],
    num ==1,
    tc = CMYKColor[1,0,0,0],
    num ==6,
    tc = CMYKColor[0,0,1,0]
   ];
   bar = {bar,CMYKColor[0,0,0,0],
      Rectangle[{xoff,num+(y-1)/numdirs},{xoff+45,num+y/numdirs}]};
   For[y = 1, y ≤ numdirs,y++,
    numpoints = Length[specie[[y]]];
    For[x = 1, x≤ numpoints,x++,
     dat = s[[y,x]]/gmaxx[[y]];
     Which[num == 5,
      color =CMYKColor[0,0+dat,0+dat,0],
      num== 4,
      color =CMYKColor[0+dat,0,0+dat,0],
      num == 3,
      color = CMYKColor[0+dat,0+dat,0,0],
      num == 2,
      color =CMYKColor[0,0+dat,0,0],
      num == 1,
      color =CMYKColor[0+dat,0,0,0],
      num == 6,
      color =CMYKColor[0,0,0+dat,0]
      ];

     bar = { bar,
```

```
          {color,
           Rectangle[{x+xoff,num+(y-1)/numdirs},{x+1+xoff,
              num+y/numdirs}]} };
       ];
      l =
       Line[{{border[[y]]+2+xoff,num+(y-1)/numdirs},{border[[y]]+2+xoff,
            num+y/numdirs}}];
      bar = {bar,{CMYKColor[0,0,0,1],l}};
      ];
    (*bar = {bar,CMYKColor[0,0,0,1]} Text[
            "r5",{border[[1]]/2+xoff,ndir+2.5}],
       Text["r4",{border[[1]]+xoff+border[[1]]/2,ndir+2.5}]};
      bar = {bar,{tc,Text[name,{-4+xoff,num+1/numdirs}]}};*)

    bar = Flatten[bar];
    bar
    ]
MakeFrame[tslice_,name_,num_,border_,ndir_,mx_,tiffs_] :=
    Module[{i,l,b = {},counter,ss,x,y,dim,g,filename},
      l = Length[tslice[[1]]];
      xoff = 6;
      For[i = 1, i ≤ l,i++,
        b = {b,
           MakeBar[SpecieSlice[tslice,i],i,name[[i]],Transpose[mx][[i]],
             border,ndir]}
        ];
      time = 7.75+Floor[num/72]*.05;
      counter = ToString[time]<>" dpc";
      x =border[[1]];
      b = {b,Text[counter,{x+xoff+3,ndir+2.5}]};
      g = Graphics[b];
      Show[g,PlotRange->All];
      If[tiffs,
        filename = ToString[num]<>".tiff";
        Display["TIFF/"<>filename,g,"TIFF",ImageResolution->300];
      ]
];

Stain[dirs:{___String},files:{___String}, start_Integer,num_Integer,
    opts___Rule] :=
  Module[{data ,name,numbers,rundat,i,j,m,l,mx={},t,mystep,sf,f = files,
      border,ndir = Length[dirs]},
    mystep=Step/.{opts}/.Options[Stain];
    sf=FrameScale/.{opts}/.Options[Stain];
    tiffs = Tiffs/.{opts}/.Options[Stain];
    l = Length[f];
```

```
data  = {};
border = {};
mx = {};
For[runs = 1, runs ≤ Length[dirs], runs++,
  AppendTo[border,ReadList[dirs[[runs]]<>"border",Real]];
  numbers = {};
   For[i = start, i ≤ num, i+=mystep,
    rundat = {};
      For[j = 1, j ≤ l, j++,
        name =dirs[[runs]]<>f[[j]]<>"."<>ToString[i]<>".dat";
        d = ReadList[name,Real,RecordLists->True];
        d = Map[Last,d];
        AppendTo[rundat,d];
        AppendTo[mx,Max[d]];
     ];
    AppendTo[numbers,rundat];
  ];
 AppendTo[data,numbers];
];
mx = Partition[mx,Length[mx]/Length[dirs]];
mx = Map[Partition[#,5]&,mx];
maximums = {};
For[i = 1, i≤ Length[mx], i++,
  AppendTo[maximums,Map[Max,Transpose[mx[[i]]]]];
  ];
For[i = 1, i <=Length[numbers],i++,
  tslice = TimeSlice[data,i];
  MakeFrame[tslice,f,i,TimeSlice[border,i],ndir,maximums,tiffs];
  ]
]
```

Plots

The plots are then invoked with the following typical commands

```
Response[{"mhoxa1","mhoxa1.100","mhoxa1.20","mhoxa1.2000","mhoxa1.7500"},5]


Movie["output.13/",{"ra","hoxa1","hoxb1","hoxb2","rar","rxr","dimer","krox20",
  "brar","brxr"},898,Step->2]

Stain[{"wt/output.13/","wt/output.17/","wt/output.19/",
  "wt/output.23/"},{"mkrox20","mhoxb2","mhoxb1","mhoxa1","ra"},1081,
 Step->1,Tiffs->True]
```

References for Appendices

Barrow, J. R., Stadler, H. S., and Capecchi, M. R. (2000). Roles of Hoxa1 and Hoxa2 in patterning the early hindbrain of the mouse. *Development* **127,** 933-44.

Gavalas, A., Studer, M., Lumsden, A., Rijli, F. M., Krumlauf, R., and Chambon, P. (1998). Hoxa1 and Hoxb1 synergize in patterning the hindbrain, cranial nerves and second pharyngeal arch. *Development* **125,** 1123-36.

Gavalas, A., Trainor, P., Ariza-McNaughton, L., and Krumlauf, R. (2001). Synergy between *Hoxa1* and *Hoxb1*: the relationship between arch patterning and the generation of cranial neural crest. *Development* **128,** 3017-3027.

Hamburger, V., and Hamilton, H. (1951). A series of normal stages in the development of the chick embryo. *J. Morph.* **88,** 49-92.

Han, K., and Manley, J. (1993). Functional Domains of the Drosophila Engrailed Protein. *Embo. J.* **12,** 2723-2733.

Itasaki, N., Bel-Vialar, S., and Krumlauf, R. (1999). 'Shocking' developments in chick embryology: electroporation and in ovo gene expression. *Nat. Cell. Biol.* **1,** E203-7.

L'Ecuyer, P., Cote, S., Brown, B. W., Lovato, J., and Russell, K. (1991). Implementing a random number package with splitting facilities. *ACM TOMS* **17,** 98-111.

LaBonne, C., and Bronner-Fraser, M. (2000). Snail-Related Transcriptional Repressors are Required in Xenopus for both the Induction of the Neural Crest and Its Subsequent Migration. *Dev. Biol.* **221,** 195-205.

Qiagen. (2000). SuperFect Transfection Reagent Handbook.

Studer, M., Gavalas, A., Marshall, H., Ariza-McNaughton, L., Rijli, F. M., Chambon, P., and Krumlauf, R. (1998). Genetic interactions between Hoxa1 and Hoxb1 reveal new roles in regulation of early hindbrain patterning. *Development* **125,** 1025-36.

Swartz, M., Eberhart, J., Mastick, G. S., and Krull, C. E. (2001). Sparking New Frontiers: Using in Vivo Electroporation for Genetic Manipulations. *Dev Biol* **233,** 13-21.

Tang, M., Redemann, C., and Szoka, F. (1996). *In vitro* gene delivery by degraded polyamidoamine denrimers. *Bioconjugate Chemistry***,** 703.

Vignali, R., Poggi, L., Madeddu, F., and Barsacchi, G. (2000). HNF1 beta is required for mesoderm induction in the Xenopus embryo. *Development* **127,** 1455-1465.

Wilkinson, D. G. (1992). Whole mount *in situ* hybridization of vertebrate embryos. *In* "In situ hybridization: A Practical Approach" (D. G. Wilkinson, Ed.), pp. 75-83. IRL Press, Oxford.