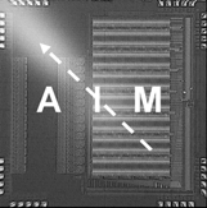


FAST: **F**actor-graph based **A**nalysis of **S**tochastic circuits

Ming Gu

*Adaptive Integrated Microsystems Laboratory,
Michigan State University*

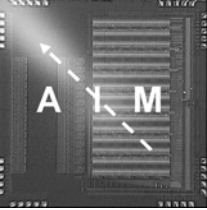
Website: <http://www.egr.msu.edu/aimlab>



Outline



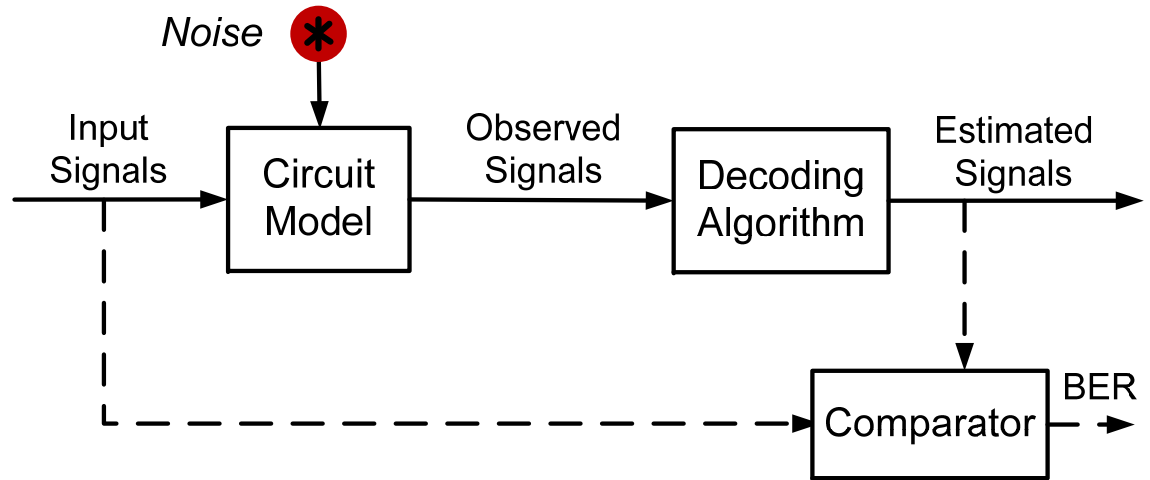
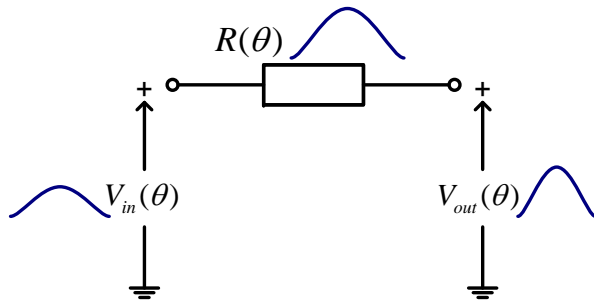
- *Motivation*
- *Tutorial*



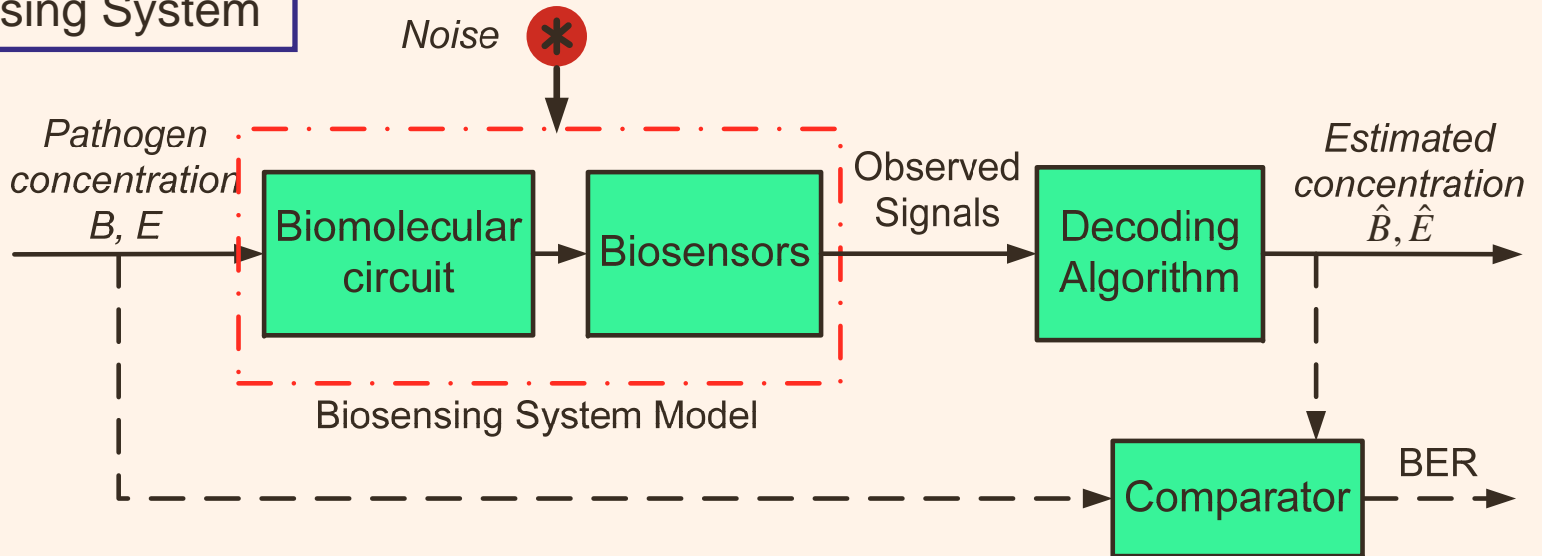
Motivation

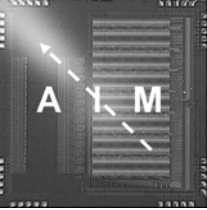


Stochastic Circuit

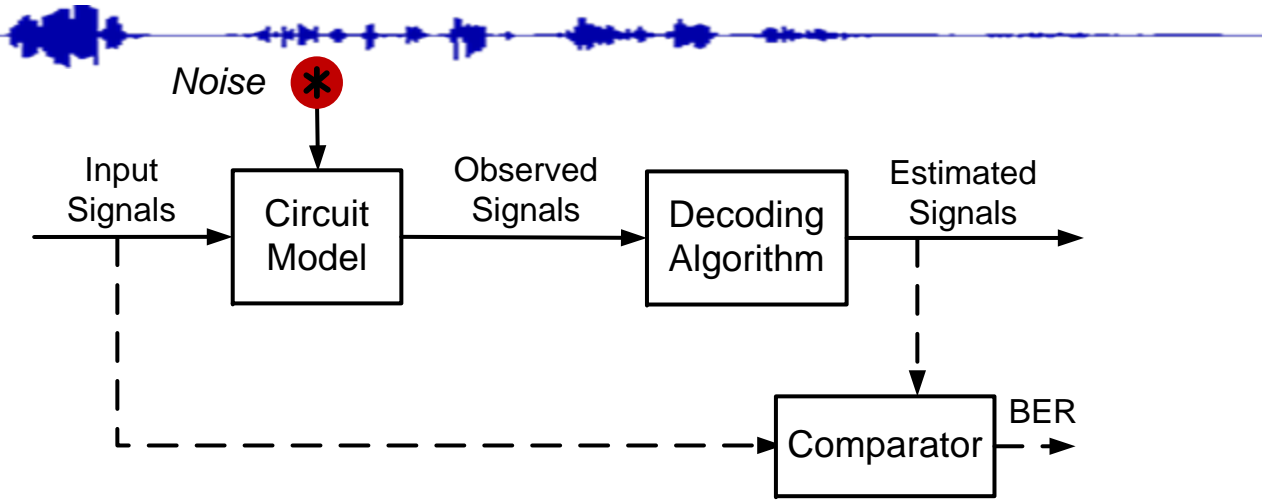


Bio-sensing System





Motivation

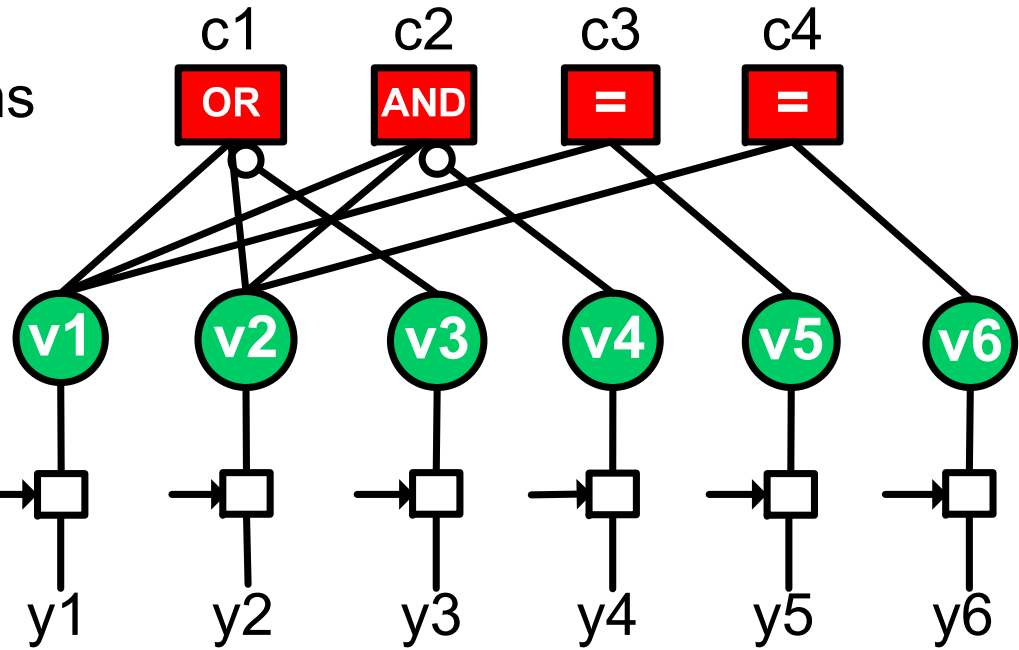


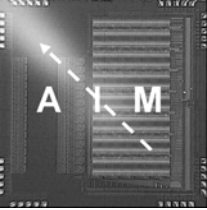
Decoding Algorithms

Observed Signals/
Estimated Signals

Noise

Input Signals



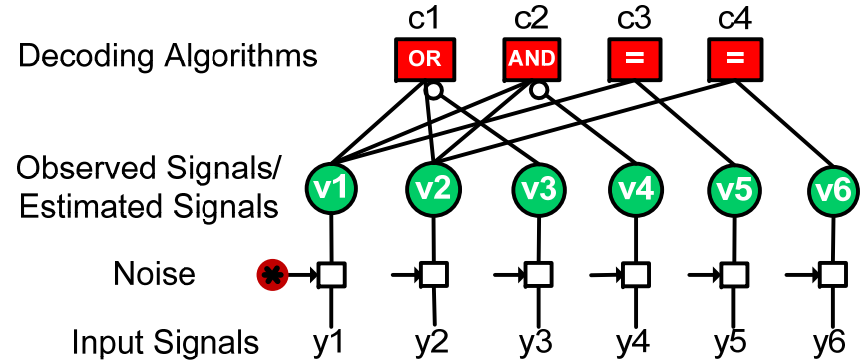


Tutorial



Step 1: Matrix Construction

Graphical Model
↓
Matrix Representation



>> `file_name="Index_BIO_IRG.dat";`

(P_Inv_main.cpp, *function main*)

>> `logic_file_name="Logic_BIO_IRG.dat";`

(P_Inv_main.cpp, *function main*)

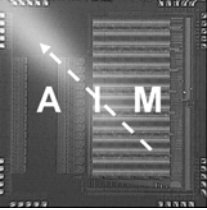
Index_BIO_IRG.dat

6 → # of variable nodes (VN)
 4 → # of check nodes (CN)
 3 → Max # of edges for CNs

1 2 3 }
 1 2 4 }
 1 5 0 }
 2 6 0 }
 Connections of c1~c4

Logic_BIO_IRG.dat

O O R → "OR"
 A A D → "AND"
 E E 0 → "Equal"
 E E 0 → "Equal"



Tutorial



Step 2: Input Signals

>> **Generate_tx.cpp,**
void cal_conductance(...)

// Parameters obtained from measured results

double Go=1.24e-6;

double kb=1.2e-6;

// Detection limit for a single biotransistor

double Xo = 0.1;

// AND and OR parameters

double Gor=13.6e-6;

double kbor=0.15e-6;

double Xobor = 0.76;

double keor=0.09e-6;

double Xeor = 8.5e-4;

double Gand = 13.1e-6;

double kband=3.4e-6;

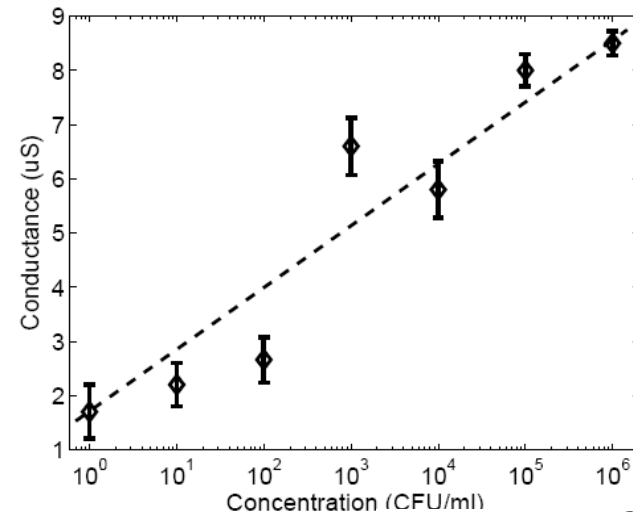
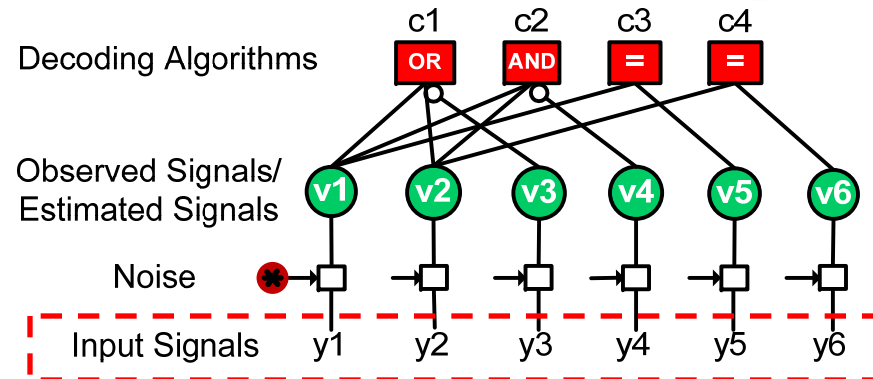
double Xoband = 1000;

double keand=0.45e-6;

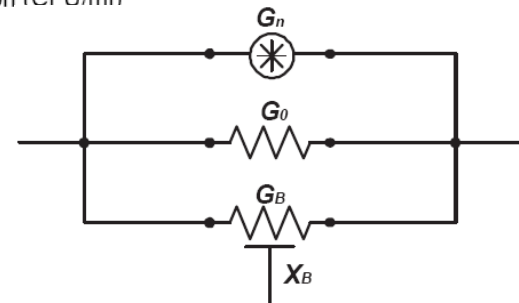
double Xeand = 460;

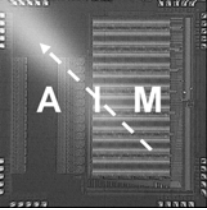
double keband=0.4e-6;

double Xeband = 1.2e3;



$$G(X_B) = G_0 + \kappa_B \log_{10} \left(\frac{X_B}{X_0} \right)$$





Tutorial



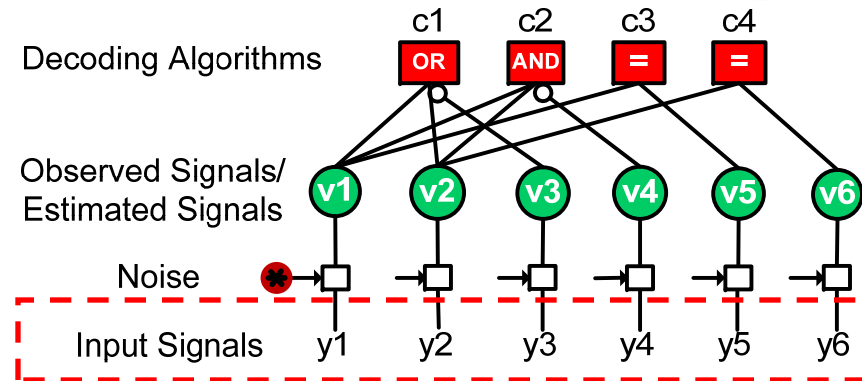
Step 2: Input Signals (cont.)

>> **Generate_tx.cpp**
void Generate_tx(...)

```
/*--- Parameters used in Normalizations ---*/  
double beta=0.2 ;  
double betaOR=0.6;  
double betaAND=0.1;
```

>> **#define CONCENTRATION_LEVEL 5**

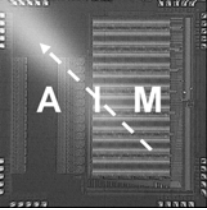
>> **double Xb=pow(10.0,i+1);**
double Xe=pow(10.0,j+1);



$$\mu_{G_k \rightarrow X_k}(1) = \frac{\exp^{\beta_k(G_k - G_0)}}{1 + \exp^{\beta_k(G_k - G_0)}}$$

(P_Inv_main.h)

(P_Inv_main.cpp, *function main*)



Tutorial



Step 3: Noise & Observed Signals

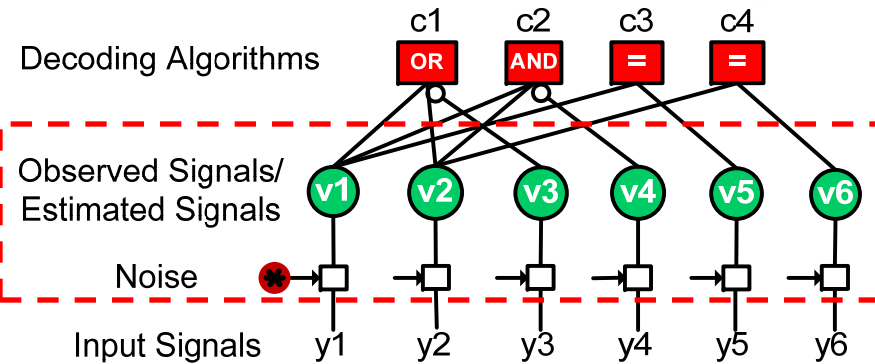
>> `add_noise.cpp`
`add_noise_bio (...)`

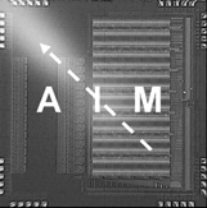
//Standard deviation obtained from fitting, combination of measurement noise and communication noise;

```
double std_Yb=3e-6;  
double std_Ye=3e-6;  
double std_Yp3=1e-6;  
double std_Yp4=1e-6;  
double std_Yp5=3e-6;  
double std_Yp6=3e-6;
```

// Parameters obtained from measured results

```
double Go=1.24e-6;  
// AND and OR parameters  
double Gor=13.6e-6;  
double Gand = 13.1e-6;
```



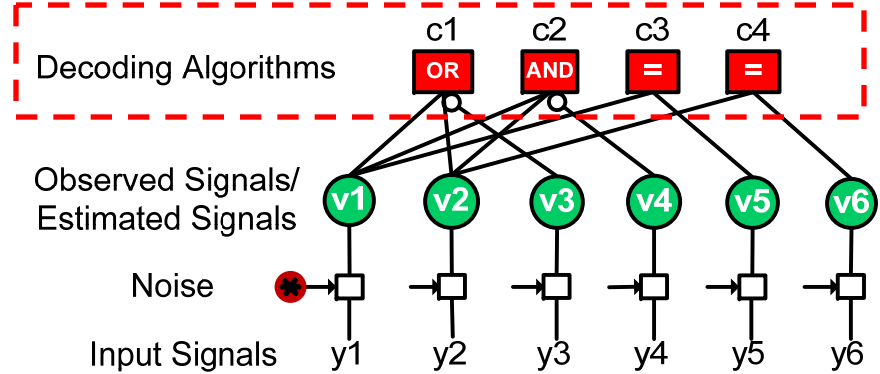


Tutorial

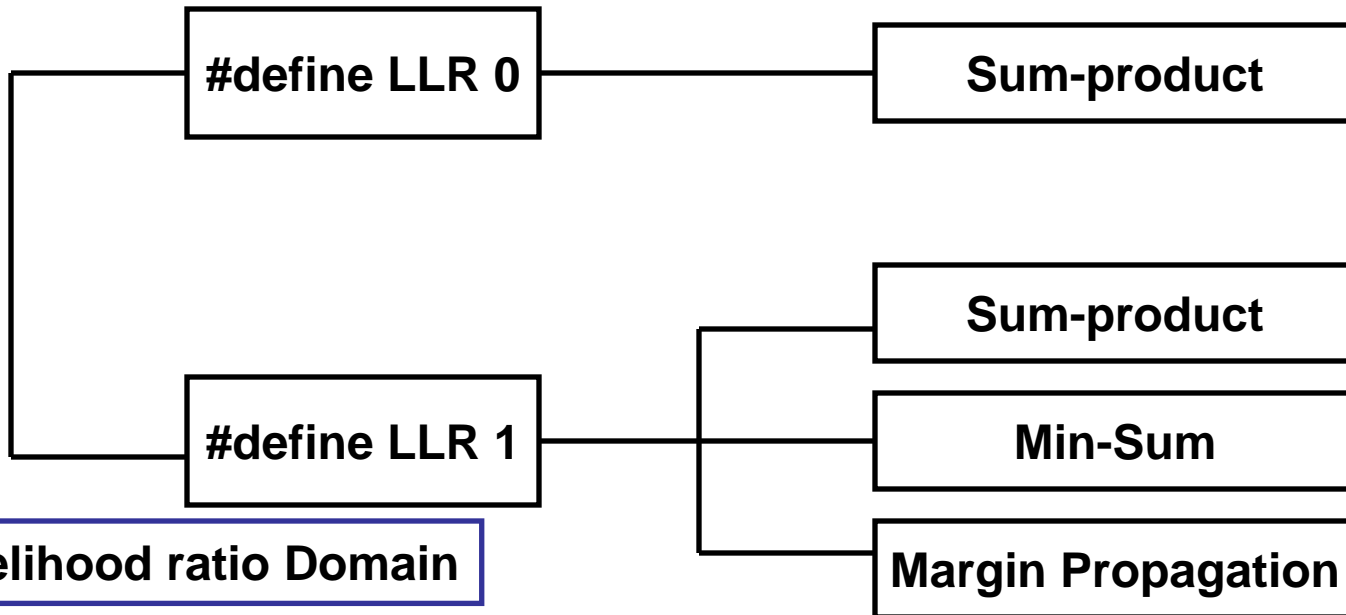


Step 4: Decoding Algorithms

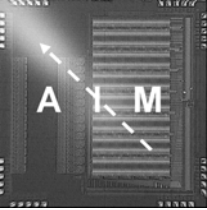
>> `#define LLR 0` (P_Inv_main.h)



Probability Domain



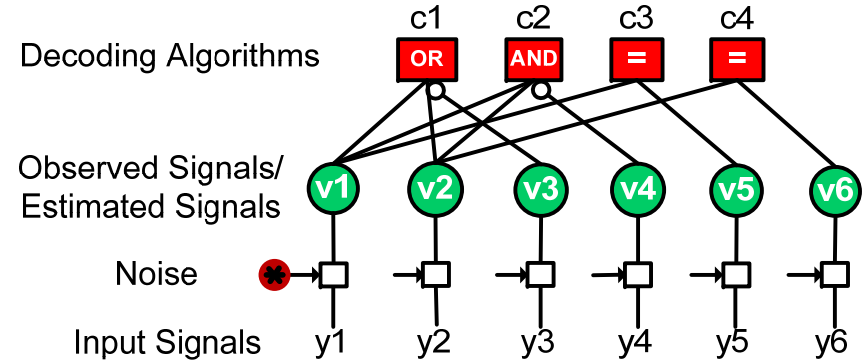
Log-likelihood ratio Domain



Tutorial



Step 5: Monte Carlo Simulation Configuration



(P_Inv_main.cpp, *function main*)

>> `int max_block=1000;`

```
while (block<max_block)
{...}
```

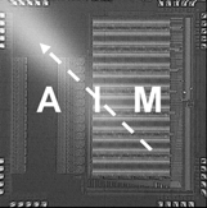
(P_Inv_main.cpp, *function main*)

>> `int max_error=4000;`

```
while (biterr_sumpro<max_error)
{...}
```

Accumulating
of blocks

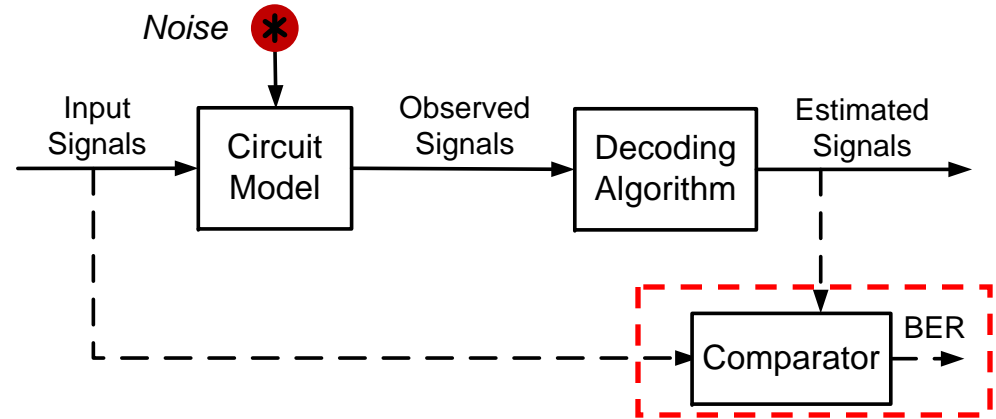
Accumulating
of errors



Tutorial

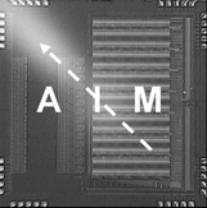


Step 6: Save a simulation output file



>> `#define SAVEFILE 1` (P_Inv_main.h)

>> `ofstream out_file("output_bio_BP.txt");` (P_Inv_main.cpp, *function main*)



Tutorial



Step 7: Run the simulation

Step 8: Check the data from the output file

