A X-Ray pixel sensor for large area imaging using VHDL-AMS

David Grant
20116363

ECE 741
Dr. Jim Barby
Department of Electrical & Computer Engineering
University of Waterloo

December 3, 2002
Preface

If this document seems unnecessarily long and wordy, I must apologize. The reason it is long and detailed is because I may attempt to publish a paper based on the results described in this document. The VHDL-AMS model will need to be advanced to a deeper level, and most likely adapted into an active-pixel sensor model. Writing an academic paper later will be easier if there is a document such as this to easily copy-and-paste the figures, equations, and detailed paragraphs from. Writing such a thorough document has also forced me to tidy certain parts of my code, and also to refine my thoughts about where to take the model next.
# Contents

1 Introduction .................................................. 5  
2 Device Operation ............................................. 7  
3 Natures .......................................................... 8  
4 Model of a Single Active X-Ray Pixel Sensor using VHDL-AMS ................................................. 9  
   4.1 Passive Pixel Sensor (PPS) ................................ 9  
   4.2 Active Pixel Sensor (APS) ................................ 10  
   4.3 X-Ray Source ............................................. 11  
      4.3.1 Simple Model ......................................... 11  
      4.3.2 Medium Model ........................................ 12  
   4.4 a-Se Solid State X-Ray Sensor ................................ 16  
      4.4.1 Simple Model ......................................... 16  
      4.4.2 Medium Model ........................................ 17  
   4.5 Thin-Film Transistor (TFT) Switch ......................... 20  
      4.5.1 Simple Model ......................................... 20  
      4.5.2 Medium Model ........................................ 20  
   4.6 Charge Amplifier .......................................... 21  
      4.6.1 Simple Model ......................................... 21  
      4.6.2 Medium Model ........................................ 22  
5 Testing and Discussion ...................................... 23  
   5.1 X-ray Source ............................................. 23  
      5.1.1 Simple Model ......................................... 23  
      5.1.2 Medium Model ........................................ 23  
   5.2 Selenium Sensor .......................................... 23  
      5.2.1 Simple Model ......................................... 23  
      5.2.2 Medium Model ........................................ 25  
   5.3 TFT .......................................................... 25  
      5.3.1 Simple Model ......................................... 25  
      5.3.2 Medium Model ........................................ 25  
   5.4 Amplifier Testing ......................................... 28  
      5.4.1 Simple Model ......................................... 28  
      5.4.2 Medium Model ........................................ 30  
   5.5 Entire System Testing ..................................... 30
5.5.1 Simple PPS Model ........................................... 30
5.5.2 Medium PPS Model ........................................ 31
5.5.3 APS Model .................................................. 33

6 Conclusion and Recommendations ................................ 36

A Code ................................................................. 37
A.1 X-Ray Source .................................................. 37
A.1.1 Entity ......................................................... 37
A.1.2 Simple Architecture ........................................ 37
A.1.3 Medium Architecture ....................................... 37
A.1.4 Test Bench ................................................... 38
A.2 Selenium Sensor ................................................ 40
A.2.1 Entity ......................................................... 40
A.2.2 Simple Architecture ........................................ 40
A.2.3 Medium Architecture ....................................... 40
A.2.4 Test Bench ................................................... 41
A.3 TFT ............................................................... 43
A.3.1 Entity ......................................................... 43
A.3.2 Simple Architecture ........................................ 43
A.3.3 Medium Architecture ....................................... 43
A.3.4 Test Bench ................................................... 44
A.4 Amplifier ......................................................... 46
A.4.1 Entity ......................................................... 46
A.4.2 Simple Architecture ........................................ 46
A.4.3 Medium Architecture ....................................... 47
A.4.4 Test Bench ................................................... 47
A.5 PPS System ....................................................... 49
A.5.1 Testbench Entity ............................................. 49
A.5.2 Medium Testbench .......................................... 50
A.6 APS System ....................................................... 52
A.6.1 Testbench Entity ............................................. 52
A.6.2 Testbench ..................................................... 52
A.7 Other ............................................................. 54
A.7.1 Resistor ....................................................... 54
A.7.2 Capacitor ..................................................... 54
A.7.3 Light Nature .................................................. 54
1 Introduction

In traditional digital radiology, amorphous silicon photo-detectors are constructed using \textit{pin} photodiodes. A phosphorous layer is placed on top of the \textit{pin} photodiodes which can detect x-rays and convert the incoming x-ray photons into photons in the visible spectrum which can then be detected by the intrinsic region of the \textit{pin} sensor. This system does not eliminate the inherently poor spatial resolution of phosphor films, which is present in phosphor screen films which have been used for a century. A different material is needed to improve the spatial resolution of a digital imager for applications such as mammography where very fine spatial resolution is needed.

Amorphous selenium films can be used to detect the x-rays directly, and convert the incoming x-rays to charge. Amorphous selenium films have a better inherent spatial resolution because they convert x-rays directly into charge which is collected at the pixel electrode. The feasibility of such a system was first proposed by Zhao and Rowlands in 1995 [1]. Their original pictures used in their original concept are shown in Figures 1 and 2.

![Figure 1: Side-view of a pixel, showing a thick a-Se region with electrodes on top and bottom, connected to a TFT for read-out.](image-url)

VHDL-AMS has been used to model passive-pixel sensors using a phosphor detection layer in the past [2]. This report will describe a newer VHDL-AMS model which simulates in detail the x-ray source, as well as the frequency dependant absorption of the amorphous selenium layer. Although this model only models one ideal pixel in an array like that shown in Figure 2, it is possible that this model could eventually be instantiated by a higher-level design to form an entire array.
Figure 2: Above view of the active matrix pixel array, showing many pixels, as well as gate lines and read-out lines.
2 Device Operation

The system is fairly simple, consisting of an amorphous selenium (a-Se) layer which absorbs x-rays and converts the x-rays into electron-hole pairs. These electron hole pairs move to opposite charged electrodes which have an electric field bias across them. The electron-hole pairs accumulate on the a-Se layer’s electrodes according to its capacitance. This capacitance is usually quite small ($\sim 0.01 \ pF$) however, so an extra storage capacitor ($\sim 1 \ pF$) is used. In a passive pixel sensor, the charge is then read-out from the pixel by switching an hydrogenated amorphous silicon (a-Si:H) thin-film transistor (TFT), which allows charge to flow into a charge amplifier which is normally implemented in crystalline silicon, external to the pixel array, and on a separate circuit. In an active pixel sensor, generated electron-hole pairs causes charge to drain off the charged capacitance of the sensor. Some amplification is also done at each pixel, as well as some external amplification, similar to passive pixel sensors.
3 Natures

In this model electrical and light natures were used. The light nature used, however, is not the same as the nature that is soon to be provided by the IEEE or by Mentor Graphics VAMS. The traditional nature would have light intensity as an across variable and light flux (in photons per second) as the through variable. When I first began my design I had flux as a non-conservative port quantity. The Mentor Graphics tool at the University of Waterloo, however, does not allow this. I found the easiest way to handle the light flux was to create a nature where flux was the across variable. The through variable would be a “dummy” variable and would be always be assigned to zero. This emulated a non-conservative port quantity in a way that made the implementation as simple as possible, by not having to worry about the light intensity across terminals.
4 Model of a Single Active X-Ray Pixel Sensor using VHDL-AMS

4.1 Passive Pixel Sensor (PPS)

The passive pixel circuit used is taken from Zhao and Rowlands [1] and is shown in Figure 3. The circuit in Figure 3 has been divided into three blocks, plus one other block which is not shown, and that is the x-ray source itself. Each block was tested individually with its own testbench, and for most of the models, multiple levels of modelling complexity were used. The most basic models are called simple, which may have some basic electrical components, however their operation is greatly simplified. The next level in the hierarchical models is called medium and these incorporate more realistic aspects of the model, although they use simplified models with many assumptions. The lowest level of complexity is advanced. None of the models are implemented yet at this level. Each of system blocks will be described individually in the sections that follow.

Briefly, the circuit operates as follows:

1. Sensing: The x-ray source is turned on, the read-out TFT is turned off, and the TFT on the feedback TFT of the op-amp is turned on. Charge accumulates on the selenium sensor’s capacitance (assuming the charge on the capacitance was already reset to 0).

2. Read-out and Reset: The read-out TFT is turned on, and the TFT on the feedback path of the op-amp is turned-off. This allows charge to be read-out from the capacitance.
ittance of the sensor, and resets the sensor at the same time, as it’s charge will go to zero.

3. Sampling: The read-out TFT is turned off, and the op-amp’s output can be sampled with an A/D, before starting this process over again.

The higher-level passive pixel sensor was easily implemented by using structural VHDL-AMS code, and by simply connecting the various modules above together. The passive pixel sensor was tested using all the simple modules from above, and then each module was changed to medium, and tested, one at a time, until the entire circuit contained only medium level models.

4.2 Active Pixel Sensor (APS)

An active pixel sensor was also implemented and briefly tested. An active pixel sensor for fluoroscopic applications has been designed by K.S. Karim [3]. The active pixel sensor which was implemented in VHDL-AMS is shown in Figure 4 below. The active pixel sensor was tested only at the medium level because the medium models have all been proven to work.

In this circuit, the operation of the pixel capacitance is opposite from that in the passive pixel sensor. In this case, the pixel capacitance is charged to the power rail voltage, and then the action of the generated electron hole-pairs is to discharge the capacitance. This is basically achieved by applying a negative voltage on the selenium layer, instead of a positive voltage in the PPS architecture above. The operation of this circuit is described as follows:

![Figure 4: Amorphous selenium active pixel sensor to be modelled. From [3]](image-url)
1. Sensing: (Assume the sensor capacitance is initially charged to the rail voltage, $V_{DD}$). The reset TFT (T3) is turned off, the amplifier reset is turned on, and the read-out TFT (T1) is turned off. The generated electron-hole pairs cause a current which discharges the pixel capacitance. This reduces the gate voltage on the gate of the amplifying TFT (T2) by an amount $\Delta V_g$.

2. Reset/read-out step 1: The amplifier reset TFT is turned off to allow the capacitor to integrate. The read TFT and the reset TFT are turned on. This produces a current through the amplifying TFT and into the external integrating amplifier. This signal will be slightly less than it would have been had there been no sensor current reducing $V_g$ by $\Delta V_g$. The read-out TFT is switched off to allow this signal to remain flat while it is stored at the first input to an external double-sampling differential amplifier.

3. Reset/read-out step 2: The integrator amplifier is reset, and then the read-out TFT is turned on again. Meanwhile, the sensor’s capacitance is now fully reset, so the current passing through the read-out TFT will be proportional to $V_g$, not $V_g - \Delta V_g$. The read-out TFT is switched off to allow this signal to remain flat while it is stored at the second input to an external double-sampling differential amplifier.

So this circuit essentially amplifies $V_g$ and $V_g - \Delta V_g$ through the amplifying TFT’s ($g_m$) transconductance ($g_m$), and the an external differential amplifier must amplify the difference between these two signals. The differential amplifier was not modelled.

### 4.3 X-Ray Source

The X-Ray sources used in medical imaging are usually made from tungsten (W) sources, or molybdenum. Traditional uses such as chest radiography, fluoroscopy, and general radiography use tungsten sources, whereas mammography uses molybdenum. X-ray sources basically emit a certain number of photons per unit time. However the spectrum of the photon energies/frequencies is spread out in frequency space as well. The simple model ignores this fact, however the medium model will take that into account. The advanced model, if implemented, would provide a more precise function of the x-ray spectrum.

#### 4.3.1 Simple Model

In the simplest model of the x-ray source, the x-ray source emits a certain amount of photons per second. This is analogous to a current source. The entity described in VHDL-AMS is shown in Figure 5 below: The simple model simply takes a generic parameter conversionFactor, which specifies how many photons are emitted per second from the
x-ray source, per unit $keVp$ of electron energy. This does not correspond directly to the input voltage, but the kinetic energy of the electrons impinging upon the metal target. This x-ray source does not have an electrical input (which would include a step-up transformer and other devices), only a generic parameter $keVp_{max}$ which is the kinetic energy of the accelerated electrons, and the maximum energy of the emitted x-rays.

### 4.3.2 Medium Model

The medium-level model incorporates the frequency dependence of the x-ray source’s output. An x-ray source operates as follows: Two electrodes are placed in a vacuum chamber, as shown in Figure 6. One of the electrodes is the source, in this case tungsten or molybdenum. A high voltage (greater than the work function of the cathode) is applied across the electrodes, which causes electrons to be ejected from the cathode with a kinetic energy equal to $\approx eV_{max}$, where $e$ is the electron charge ($1.6 \times 10^{-19}$). The $V_{max}$ voltage is normally measured in $kV$, and the electrons emitted are said to have an energy in units of $keVp$. These electrons escape the cathode and hit the anode, which is our source metal. The electrons which do not actually collide with the atoms in the source are decelerated by the heavy atoms in the source, and as they decelerate they emit their energy in the form of high-energy x-rays. This type of radiation is called Bremsstrahlung radiation. Many of the low energy x-rays ($< 5 keVp$) are absorbed in the source itself as they try to escape. The higher energy x-rays escape, and these are the useful x-rays which are emitted from the x-rays tube. An example of a tungsten x-ray spectrum is shown in Figure 7. This graph is
Figure 6: Simple X-ray tube

Figure 7: Example tungsten x-ray spectrum. From [4]
from Tucker and Barnes [4], who did some important work on modelling the spectrum in 1991. This spectrum shown in Figure 7 does model accurately the absorption and attenuation of the x-ray photons in the source material, however it does not include other external effects such as filtering at the output of the x-ray tube, which is common in modern x-ray tubes, to fine-tune the spectrum for application to medical imaging. Dr. I. A. Cunningham et al. from the University of Western Ontario took the models from Tucker and Barnes and made a C program called \texttt{xspec.c} which calculates the x-ray spectrum of a tungsten or molybdenum source in 1995. The \texttt{xspec.c} program allows the user to input the target type, emission angle, Pyrex thickness, oil thickness, lexan thickness, aluminium thickness, molybdenum thickness, electrode voltage, current, distance to radiographic screen, starting photon energy (about 5 keVp), energy spacing, and number of bins. In other words, a complete description of the x-ray source in question can be provided to the model. Liz Yang produced a Windows mexxspec.dll file for use with Matlab in 1999 [5]. I created a Matlab script to execute this mexxspec.dll file and parse the output array into a Mathematica array format. For this project, I simulated a tungsten x-ray tube, the “GE tube Maxiray 125” at 80 keVp, using the following lines of Matlab code:

\begin{verbatim}
% MEXXSPEC(desc, target, degrees, mmpyrex, mmoil,
% mmlexan, mmAl, mmMo, kVp, mAs, mdis, E0, dE, fnbins)
[phi R] = MEXXSPEC('GE tube Maxiray 125',2,10,2.38,3.06,
2.66,1.5,0,80,30,1,5,1,81)
\end{verbatim}

The data produced by \texttt{xspec.c} is very close to the output of true x-rays sources, and has been perfected and tweaked over many years in order to achieve this. The spectrum produced by the Matlab script will act as the experimental data for the purposes of the VHDL-AMS model. A plot of this data is shown in Figure 8.

Using Mathematica, I was able to fit a polynomial accurately to the spectrum produced in Matlab. In Mathematica, fitting non-linear functions with many parameters to a set of data is generally easy. However, on the first attempt, I tried to fit multiple Gaussians, but this was unsuccessful. I found that a sixth-order polynomial was the next best thing for simulating this x-ray spectrum. The fit polynomial is shown in Figure 9. Although the peak of the spectrum is not quite centered where it should be, it still retains the overall shape, and it very suitable for the medium architecture. The polynomial that Mathematica fit to the model is as follows,

\begin{equation}
\Phi_0 = -0.00152138E^6 + 0.199592E^5 + 5.62405E^4 - 1794.43E^3 + 77635E^2 - 825540E + 1.79835 \times 10^6
\end{equation}
Figure 8: Spectrum obtained from xspec.c

Figure 9: Sixth-order polynomial fit to spectrum in Figure 8 in Mathematica
The x-ray source in the medium architecture must produce the x-ray output between 0 keV and the maximum energy (in this case 80 keV) within some given time frame. This time frame begins when the digital enable input signal goes high and ends when the shutterTime has expired. So the frequency sweep can occur over any time duration. Time is mapped into the energy \( E \) (called current_keV in the code) by the following code:

\[
current\text{._keV} = \frac{((\text{now} - \text{time2real(start\_time)}) \times \text{keVp\_max})}{\text{time2real(shutter\_Time)}};
\]

The start\_time signal is set by the following process:

```
timer : process (enable)
begin
    if enable 'event and enable = '1' then
        start\_time <= real2time(now);
    end if;
end process timer;
```

### 4.4 a-Se Solid State X-Ray Sensor

The amorphous selenium layer absorbs x-rays and converts a certain number of them into a certain number of charges at each pixel. This conversion is dependant on the frequency of the incoming x-rays. The simple architecture assumes no frequency dependency in the a-Se film, and the medium architecture uses the x-ray frequency dependant medium architecture of the x-ray source, and converts the photons into energy as a function of frequency/energy.

#### 4.4.1 Simple Model

The selenium x-ray sensing film entity is shown in Figure 10. There is a light flux input to the selenium sensor entity. Notice that as described in section 3 above, the flux is an across quantity. The selenium sensor block does not “draw” any flux as an electrical circuit draws current. It simply “observes” or “senses” this flux and produces an appropriate current. The pixel capacitance, given by the sum of \( C_{Se} \) and \( C_{st} \) in parallel, is included in the selenium sensor model. It makes sense to include these capacitances in the sensor’s model, because, in fact the \( C_{Se} \) element is the capacitance of the selenium itself. This simple architecture has a photocurrent \( I_{ph} \) which is proportional to the incoming flux by some constant, which is called conversion_factor. The current and voltage at the output terminal are determined from the following code:

```
if domain = quiescent\_domain use
    v\_out == 0.0;
else
    i\_out == \text{conversion\_factor} \times \text{flux} \times q + \text{Cpix} \times v\_out\_dot;
end use;
```
4.4.2 Medium Model

The absorption of x-rays in the a-Se film is actually frequency dependant. Since a frequency dependant x-ray source has been modelled, this is useless without making the sensor frequency dependant as well.

The absorption is going to be a function of the number of photons at a specific energy as well as the energy of those photons. The incident x-ray flux will be given by $\Phi(E)$. It also depends on the parameters of the pixel, the incident flux, and the quantum efficiency, which is given by

$$\eta(E) = 1 - e^{-\mu(E)d_{Se}},$$  \hspace{1cm}(2)$$

where $d_{Se}$ is the thickness of the a-Se layer and $\mu(E)$ is the linear x-ray attenuation coefficient in Se. Since not all attenuated x-ray energy is absorbed by a-Se, the quantum efficiency is multiplied by the ratio of $\frac{\mu_{ab}(E)}{\mu(E)}$ where $\mu_{ab}$ is the absorption coefficient of a-Se. What we end up with is an equation for the charge on one pixel,

$$Q_s = \int_{0}^{E_{max}} \frac{\Phi(E)\eta(E)A_PF_P \mu_{ab}(E)}{W_{\pm} \mu(E)} EdE$$  \hspace{1cm}(3)$$

where $W_{\pm}$ is the energy (in keV) required to excite one electron-hole pair (think of this as the bandgap), $F_P$ is the pixel fill-factor, and $A_P$ is the total pixel area. The equation above is transformed into the time domain, because Mentor Graphics ADMS only allows integration
in the time domain. The photo current is thus computed from

\[ I_{ph} = \frac{dQ_s}{dt} = \frac{dQ_s E_{max}}{dt \ t_{max}} = \frac{\Phi(E)\eta(E)A_P F_P \mu_{ab}(E)\ W_\pm}{\mu(E)} E E_{max} \ t_{max} \]

(4)

This was coded in VHDL-AMS as follows:

```vhdl-ams
if domain = quiescent_domain use
    sensor_current == 0.0;
    v_out == 0.0;
else
    sensor_current == (flux*sensor_area*
                        quantum_efficiency*absorption_ehp*
                        fill_factor*keVp_max*current_keV*q) /
                        (W*absorption_total*time2real(shutterTime));
    i_out == sensor_current + Cpix*v_out 'dot;
end use;
```

`current_keV` is determined in the same way as for the x-ray sensor architecture. The only thing left to determine is the energy dependant absorption coefficients of selenium, which are shown in the code block above. These were found from Photocoef [6] on the Internet. The raw data was then fit in Excel. The absorption coefficients are fit nicely with a power relation and these are shown in Figures 11 and 12. The quantum efficiency is fit well with a third-order polynomial, and is shown in Figure 13. Further improvements can be made in a more advanced model. The best approach would be to use the actual experimental data, and to simply interpolate linearly for points in between.
Figure 12: Energy-dependant photoelectric absorption, $\mu_{ab}$ for selenium

Figure 13: Energy-dependant quantum efficiency, $\eta$ for selenium
4.5 Thin-Film Transistor (TFT) Switch

Thin-film transistors used in x-ray sensors are normally made of a-Si:H although CdSe TFTs have been used in the past [7]. For this VHDL-AMS, a single TFT will be used to read-out the charge on the gate. This will be a passive-pixel architecture since no amplification will be done in the pixel region. The TFT will just be acting as an on-off switch. TFTs are field-effect devices, similar to MOSFETs, only their mobility is much lower and hence driver much lower currents, and have high on and off resistances.

4.5.1 Simple Model

The simplest model models the transistor as a variable resistance, controlled by the gate switch. When the gate voltage is above the threshold voltage, \( V_t \), there will be a resistance of \( R_{on} \) between the source and drain terminals. When the gate voltage is below \( V_t \), there will be a resistance of \( R_{off} \) between the source and drain terminals. This discontinuous behaviour in the output current when the TFT is switched on or off can cause problems in the analog solver, so a break on \( v_{\text{gate}} \) above \( (V_t) \); command is used to alert the solver when the discontinuity occurs. A block diagram of the TFT entity is given in Figure 14.

\[
\begin{align*}
\text{TFT entity} & \\
\text{drain} & \text{source} \\
\text{gate} & \\
+ & - \\
V_t & \\
\end{align*}
\]

Figure 14: Block diagram of TFT entity

4.5.2 Medium Model

A more advanced model of the TFT can be found by considering a simple co-planer TFT, like that shown in Figure 15.

The model used is taken from [8] and it assumes the gradual channel approximation applies. It can easily be shown that if the \( t_a \) is fairly small, the current in the channel is given by

\[
I_d = \mu_n C_i (W/L) \left[ (V_g - V_t)V_d - \left( \frac{V_d^2}{2} \right) \right],
\]

(5)
where $V_t = -e \tau_d n_0 / C_i$ and $C_i = \epsilon_i / t_i$. Clearly, a-Si:H TFTs operates like a depletion-type field-effect transistors, with a negative threshold voltage. This is only valid in the triode region of operation, for $0 < V_d < V_g - V_t$. In the saturation regime, the current is given by

$$I_{d,sat} = \frac{1}{2} \mu_n C_i (W/L) (V_g - V_t)^2$$

for $V_d > V_g - V_t$. This is modelled in VHDL-AMS easily by using various if...use...else statements.

### 4.6 Charge Amplifier

The charge amplifier is just a simple model, and not much emphasis was put into modelling it. It is made up of an op-amp in a simple charge integrator configuration whose output can be reset by shorting out the feedback loop. This is normally done with a MOSFET; however, for the medium architecture, I have used a TFT as the reset switch instead. The basic amp entity is shown in Figure 16.

![Figure 16: none](image-url)
4.6.1 Simple Model

The simple architecture simple uses a different feedback resistance depending on the value of the reset voltage. If the voltage is greater than 0.0, the reset is considered active, and the feedback path is short-circuited. If the voltage is less than 0.0, the reset is disabled, and the feedback path is high-impedance, so only the feedback integrating capacitor conducts.

4.6.2 Medium Model

The medium model uses a TFT as the switch element in the feedback path. The TFT acts in a similar way to the variable resistance used in the simple model. When the TFT is turned on with a gate voltage greater than its threshold voltage, it has a resistance $R_{on}$. Conversely, when the TFT is turned off, it has a resistance of $R_{off}$. So there is not much difference between the medium and simple integrator models.
5 Testing and Discussion

5.1 X-ray Source

5.1.1 Simple Model

The x-ray source simple model was tested first. This just involved switching on and off the enable switch to make sure the output flux was turned on and off properly. This is shown in Figure 17.

![Figure 17: Test bench waveforms for simple x-ray source model](image)

5.1.2 Medium Model

The x-ray source medium model is much more complicated than the simple model. Looking at Figure 18 however, it appears to work perfectly. After the enable signal goes high, the x-ray source’s spectrum is supposed to sweep in the energy domain up to 80 keV, which is does, according to the current_keV curve. The flux_neg curve is also properly rectified to produce the flux curve.

5.2 Selenium Sensor

5.2.1 Simple Model

There is really not much to testing the simple selenium model. There are only two parameters, the sensor’s capacitance which is 0.01 pF, and the extra storage capacitance, which is 1 pF. However, it is provided for completeness, and shown in Figure 19. The input flux produces a voltage on the sensor’s capacitance, and this behaviour is seen in the graph.
Figure 18: Test bench waveforms for medium x-ray source model

Figure 19: Test bench waveforms for simple selenium sensor model
5.2.2 Medium Model

The medium model takes into account a few parameters of the selenium sensors. The following parameters shown in Table 1 were used in the testbench for the medium selenium sensor.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{st}$</td>
<td>$1.0 \times 10^{-12}$</td>
<td>pF</td>
</tr>
<tr>
<td>$C_{Se}$</td>
<td>$0.01 \times 10^{-12}$</td>
<td>pF</td>
</tr>
<tr>
<td>$keV_{p_{inin}}$</td>
<td>5.0</td>
<td>keV</td>
</tr>
<tr>
<td>$keV_{p_{max}}$</td>
<td>80.0</td>
<td>keV</td>
</tr>
<tr>
<td>shutterTime</td>
<td>1.0</td>
<td>µs</td>
</tr>
<tr>
<td>sensorArea</td>
<td>$0.25 \times 0.25$</td>
<td>mm$^2$</td>
</tr>
<tr>
<td>$W_{\pm}$</td>
<td>0.087</td>
<td>keV</td>
</tr>
<tr>
<td>fillfactor</td>
<td>0.85</td>
<td>no units</td>
</tr>
</tbody>
</table>

The medium model adds the x-ray frequency dependant nature of the selenium material. It works much like the x-ray source medium model. When the enable signal (not shown) goes high, the sensor expects to see a flux of photons at the $current_{keV}$ energy on its flux input. As the $current_{keV}$ increases, the model constantly recomputes the absorption coefficients and the quantum efficiency. This can be seen in Figure 20. It is interesting to note that the sensor current seems to increase, even though the absorption and quantum efficiency is decreasing! This is due to the fact that the $current_{keV}$ is increasing, so x-rays at these energies produce many electron hole pairs even though the absorption is much lower than at low energies. In the bottom graph in Figure 20, the voltage across the sensor’s capacitance can be seen as it charges to about $-10$ V (a non-realistic value, because a true x-ray spectrum was not used in this testbench).

5.3 TFT

The TFT was tested by creating $I_{ds}$ vs. $V_g$ curves, as well as $I_{ds}$ vs. $V_{ds}$ curves.

5.3.1 Simple Model

For the simple model, only $I_{ds}$ vs. $V_{ds}$ curves were used, since the $I_{ds}$ vs. $V_{g}$ curves would be trivial. These curves are shown in Figure 21.

5.3.2 Medium Model

A fairly standard technology TFT was instantiated in the testbench. The values for the generic parameters used are shown in Table 2. They were taken from an example given in
Figure 20: Test bench waveforms for medium selenium sensor model
The $I_{ds}$ vs. $V_g$ characteristics are shown in Figure 22. There is no experimental data to verify the above model. Since no detailed model of the leakage current was used, comparing figures of merit such as the on-off ratio would be a useless exercise anyways. The on-current of $10\mu A$ at $V_d = 10\ V$ in this VHDL-AMS model is consistent with common TFT designs, however, and it has threshold voltage of about $-1\ V$ which is to be expected, given the standard TFT parameters used.

The $I_{ds}$ vs. $V_{ds}$ characteristics are shown in Figure 23. As seen in the figure, the model properly transitions from the triode region into the saturation without any discontinuity.
The TFT exhibits good transistor action, although it is quite ideal. A more complex model will be needed in a more advanced version of this model.

5.4 Amplifier Testing

The amplifier was tested with a voltage source, input resistance, and output load. The gain should be equal to the feedback resistance over the input resistance, since this is an ideal op-amp connected with negative feedback. So the gain should be given by,

\[ G = \frac{1}{2\pi f R_{in} C_{int}} \] (7)

For a \( f = 1 \, kHz \) sine wave, \( R_{in} = 20 \, k\Omega \), and \( C_{int} = 5.0 \, pF \), the gain should be 150.

5.4.1 Simple Model

The waveforms from the testbench for the simple model are shown in Figure 24. There is clearly a gain of about 150 from \( v_{source} \) to \( v_{output} \), and the input signal, a cosine wave, has been integrated and negated into a negative sine wave. The reset function also works, shorting out the feedback path and discharging the capacitor.
Figure 23: Test bench waveforms for medium TFT model under varying drain voltage for different gate voltages
5.4.2 Medium Model

The medium model’s behaviour is similar, only there is a little bit more leakage, since the off-
resistance of the TFT is not quite high enough to completely remove the capacitor’s effect. It’s testbench waveforms are shown in Figure 25. This also shows a gain of 150 and proper functionality of the reset line.

5.5 Entire System Testing

After testing all of the models at the simple and medium level, the next step is to combine all the models into pixel design and confirm that it operates at expected. A passive pixel sensor (PPS) circuit, like that shown in Figure 3 was first tested. Provided there was enough time, a more complex (3 TFTs instead of just 1) active pixel sensor (APS) would be attempted as well.

5.5.1 Simple PPS Model

The testbench for the passive pixel was created by attaching together and x-ray source, a selenium sensor, a read-out TFT, and an integrating amplifier. For this simple model, only the simple architectures were used for all the components. The simple model is shown

Figure 24: Test bench waveforms for simple integrator model
Figure 25: Test bench waveforms for medium integrator model

below in Figure 26. The first graph shows a constant flux created from the x-ray source. The second graph is the \( v_{\text{out}} \) from the sensor, showing the increasing voltage on the sensor’s capacitance. The third graph shows the current through the read-out TFT, which is initially zero, but increases and then decays once the read-out TFT is switched on by the \( v_{\text{gate}} \) signal, seen in the fourth graph. The fifth graph is of the voltage output from the amplifier, which begins at zero, and decreases to about -18mV (this voltage is negative due to the inverting nature of the amplifier in the negative feedback mode). The last graph shows the reset line of the op-amp, which is turned off at the same time that the read-out TFT is turned on, to allow charge to accumulate on the op-amp’s feedback capacitor. This circuit behaves as expected.

5.5.2 Medium PPS Model

For the medium level testing, all the individual components were taken from the medium level. This system was tested gradually, by changing one element at a time into medium level architectures, until all the elements were at the medium level. The testbench results for the medium model are identical to those obtained for the simple model, except that the x-ray spectrum is now more realistic, and energy dependant, the selenium sensor is more realistic,
Figure 26: Test bench waveforms for simple PPS
the TFT is now an actual non-linear device, and the amplifier is pretty much the same as in the simple model. The testbench waveforms are shown in Figure 27. In order to show that the shutterTime parameter of the x-ray source and the selenium sensor does not affect the amount of current in the sensor, the shutterTime was varied, and in fact, the voltage accumulated on the sensor’s capacitance remained the same in all cases. So the amount of photo current, and hence voltage on the sensor capacitance is proportional only to the amount of photons, and the energy range of the photons (defined by keV_{\text{max}}).

5.5.3 APS Model

The active pixel model was easily made, simply by adding a $V_{DD}$ power supply, as well as 2 TFTs, and connecting them appropriately into the circuit. The results are given in Figure 28. It can be seen in the graph of $v_{\text{gate}}$ that the incoming flux on the sensor, causes the voltage on the gate of the amplifying TFT (T2) decreases slightly. (This voltage $v_{\text{gate}}$ is relative to the power rail). This drop in the gate voltage of T2 is removed once, the reset TFT is turned on again, and the capacitor is re-charged. The read-out TFT was sampled twice, as seen in the $v_{\text{read\_gate}}$ waveform. This produced two voltage ramps at the output, as seen in $v_{\text{output}}$ waveform (the uppermost graph). These two voltages can be sampled and then compared, by an external double-sampled differential amplifier.

Not much testing was done on this circuit, which is probably why there is not much gain, which there should be. In hindsight, I think the main reason that there isn’t enough gain is that the bias voltage was not high enough, and that the transistor’s (W/L) factor was not high enough. Also, the extra storage capacitance, $C_{st}$ value has a large effect as well. The two output voltage ramps ($\sim 0.20 V$) have a higher voltage than in the passive pixel sensor ($\sim 0.07 V$), however, the difference between these two waveforms is smaller ($0.05$). So it would appear that the active pixel sensor is worse than the passive pixel sensor. However, the design needs to be optimized, and also it needs to be tested using different input radiation levels (for example through bone or skin), to see the linearity of the two models.
Figure 27: Test bench waveforms for medium PPS
Figure 28: Test bench waveforms for APS
6 Conclusion and Recommendations

It is clear that VHDL-AMS can emulate the energy domain for an x-ray source and sensor, if the time domain used instead, and the proper conversions are done. The time needed in the time-domain for the energy sweep is up to the user, and is completely arbitrary. Non-linear fits were made to the absorption spectrum of selenium, as well as the x-ray output of a typical x-ray tube. There were no problems with simulating this in VHDL-AMS. A basic TFT model was also created, and worked perfectly, although the model was somewhat ideal. The passive pixel and active pixel architectures work as intended, and there were no problems with the simulation.

The next step should be to decide which models need enhancement, and which do not. The medium level models already took significant time to simulate (tens of seconds), so care should be used in creating more advanced models, that the CPU use does not increase unnecessarily. The x-ray source should be improved. Two polynomials should be used, one for the low-energy tail, and the other for the high-energy tail. This will make the x-ray source’s peak line-up more with the true peak location. This will probably change the sensor output quite a bit. More x-ray sources could also be programmed into the x-ray source. Currently it is hard-wired to the 80 keV source. It would even nicer if one could call the xspec.c file from VHDL-AMS to find the spectrum for any source. If not, a perl script run just before compiling could do this, inserting a code block for the chosen x-ray source into the x-ray source architecture. A better fit to the data is needed for the selenium sensor’s absorption. Multiple fits could be used, for different energy ranges. This will slow down the simulator somewhat, as it will have to slow down at the discontinuities. A more advanced TFT is probably critical, especially for the amplifying TFT in the active pixel architecture, since it uses a continuous range of gate voltages as well as drain-to-source voltages. Lastly, a “body” block should be made, which is similar to a transmission line in communication systems. This “body” block will emulate the human body, and can vary between bone and fatty tissue. This will act similar to the selenium sensor, as it will have an energy dependant absorption spectrum.

VHDL-AMS has proven to be an adequate simulator for the task of simulating a thin-film selenium x-ray sensor. As described above, the model could be expanded even further, if expanded to a sufficient level, could help in the design and optimization of x-ray sensors for diagnostic medical applications.
A Code

A.1 X-Ray Source

A.1.1 Entity

library disciplines, work;
use work.radiant_systems.all;
use disciplines.electromagnetic_system.all;

definition xraySource is

generic
— for simple model
conversionFactor : real := 2.5e5; — Simple conversion factor
keVp_max : real := 80.0; — Maximum photon energy

— for medium model
shutterTime : time := 1 us; — The amount of time the x-ray source
— should be shuttered.
keVp_min : real := 5.0; — Minimum photon energy
exposure_matlab : real := 1.0 — Exposure (from Matlab)

— for advanced model
)

port
— Digital
enable : in bit; — This is like an on-off switch

— Analog, light nature
terminal lightOutput : radiant; — xray output terminal (the xray flux
— appears ACROSS this terminal). I know
— this in non-standard, but I just wanted
— a port quantity.

  terminal ref_radiant : radiant
);

begin
assert keVp_max < 1000.0
report "Check units of keVp_max. It should be specified in keV, not eV"

severity warning;
end entity xRaySource;

A.1.2 Simple Architecture

architecture simple of xraySource is

quantity flux across nothing through lightOutput to ref_radiant;

begin
if enable = '0' use
flux == 0.0;
else — enable = '1'
— flux output in photons/mn^2/mR
flux == conversionFactor*keVp_max;
end use;

break on enable;
end architecture simple;

A.1.3 Medium Architecture

library MGC_AMS, ieee;
use MGC_AMS.conversion.all;
use ieee.math_real.all;

37
architecture medium of xraySource is
    quantity flux across nothing through lightOutput to ref_radian;
signal start_time : time := 0.0 ns;
quantity current_keV : real := 0.0;
quantity flux_neg : real := 0.0;
begins
  -- Set the current time to be the start_time
  timer : process (enable)
    begin
      if enable'event and enable = '1' then
        start_time <= real2time(now);
      end if;
    end process;

  -- flux output in photons/mm²/keV/R
  if current_keV > keVp_min and current_keV < keVp_max and enable = '1' use
    flux_neg == (+1.79835e6 − 825540.0*current_keV + 77635.0*(current_keV**2)
                −1794.43*(current_keV**3) + 5.62405*(current_keV**4)
                + 0.199592*(current_keV**5) − 0.00152138*(current_keV**6))/exposure_matlab;
  else
    flux_neg == 0.0;
  end use;

  -- This part acts as a rectifier
  if flux_neg < 0.0 use
    flux == 0.0;
  else
    flux == flux_neg;
  end use;

  -- Translation from time to energy
  current_keV == ((now−time2real(start_time)) * keVp_max)/time2real(shutterTime);
break on enable;
break on current_keV'above(keVp_min);
break on flux_neg'above(0.0);
end architecture medium;

A.1.4 Test Benches

library disciplines,work.MGCAMS;
use work.all;
use MGCAMS.conversion.all;
use work.radiant_systems.all;
use disciplines.electromagnetic_system.all;

entity xraySource_bench is
generic(
  init_delay : time := 1 ns
);
end entity xraySource_bench;

architecture bench_simple of xraySource_bench is
  signal enable : bit := '1';
  terminal light_sensor : radiant;
begins
  turn_on_and_off : process (enable)
    begin
      turn_on_and_off enable <= not(enable) after 20 us;
    end process turn_on_and_off;

dut : entity xraySource(simple)
generic map(keVp_max => 80.0, conversionFactor => 2.5e5)
port map(enable => enable, lightOutput => light_sensor, ref_radian => radiant_ref
architecture bench_simple is
begin
  enable := '0';
  light_sensor := radiant;
  enable <= '1' after 1 us, '0' after 10 us, '1' after 11 us;
  dut : entity xraySource (medium)
    generic map (shutterTime => 7 us,
                 keVp_max => 80.0,
                 keVp_min => 5.0,
                 exposure_matlab => 1.249)
    port map (enable => enable,
              lightOutput => light_sensor,
              ref_radiant => radiant_ref);
end architecture bench_simple;

architecture bench_medium of xraySource_bench is
  signal enable : bit := '0';
  terminal light_sensor : radiant;
begin
  enable <= '1' after 1 us, '0' after 10 us, '1' after 11 us;
  dut : entity xraySource (medium)
    generic map (shutterTime => 7 us,
                 keVp_max => 80.0,
                 keVp_min => 5.0,
                 exposure_matlab => 1.249)
    port map (enable => enable,
              lightOutput => light_sensor,
              ref_radiant => radiant_ref);
end architecture bench_medium;
A.2 Selenium Sensor

A.2.1 Entity

```vhdl
library disciplines, work;
use work.radiant_systems.all;
use disciplines.electromagnetic_system.all;

entity seleniumSensor is
  generic
    —for all models
    Cst : real := 1.0e-12; —Capacitance of storage capacitor (in Farads)
    Cse : real := 0.01e-12; —Capacitance of selenium (in Farads)
    —for simple model
    conversion_factor : real := 10000.0; —# of electrons you get for every photon
    —for medium model
    keVp_min : real := 5.0;
    keVp_max : real := 80.0;
    shutterTime : time := 1.0 us;
    sensor_area : real := 0.25*0.25; —area of sensor in millimeters
    W : real := 0.087; —energy needed to generate EHP (in keV)
    fill_factor : real := 0.85; —sensor fill factor (effective fill factor)
    exposure : real := 1.0e-3 —exposure in Roentgens
    —for advanced model
  end generic;

port(
  enable : in bit;
  terminal lightInput : radiant;
  terminal lightRef : radiant;
  terminal output : electrical;
  terminal ref : electrical);

begin
  dummy == 0.0;
  if domain = quiescent_domain use
    v_out == 0.0;
  else
    i_out == —conversion_factor * flux * q + Cpix*v_out’dot;
  end use;
end entity seleniumSensor;
```

A.2.2 Simple Architecture

```vhdl
architecture simple of seleniumSensor is
  constant q : real := 1.6e-19; —electron charge
  constant Cpix : real := Cst + Cse; —total pixel capacitance

quantity flux across dummy through lightInput to lightRef;
quantity v_out across i_out through output to ref;

begin
  dummy == 0.0;
  if domain = quiescent_domain use
    v_out == 0.0;
  else
    i_out == —conversion_factor * flux * q + Cpix*v_out’dot;
  end use;
end architecture simple;
```

A.2.3 Medium Architecture

```vhdl
library MGCAMS, ieee;
use MGCAMS.conversion.all;
```

40
use ieee.math_real.all;

architecture medium of seleniumSensor is

constant q : real := 1.6e−19;  -- electron charge
constant Cpix : real := Cst + Cse;  -- total pixel capacitance

begin
  dummy == 0.0;

  if enable ’event and enable = ’1’ then
    start_time <= real2time(now);
  end if;

end process;

begin
  timer : process (enable)
  begin
  if enable ’event and enable = ’1’ then
    start_time <= real2time(now);
  end if;
  end process;

  if current_keV > keVp_min and enable = ’1’ use
    quantum_efficiency == 1.0e−6*(current_keV**3.0)
    −0.0002*(current_keV**2.0)
    −0.0021*(current_keV)
    +1.0099;
    absorption_ehp == 17020.0*(current_keV**(-1.7039));
    absorption_total == 17268.0*(current_keV**(-1.6699));
  else
    quantum_efficiency == 0.0;
    absorption_ehp == 0.0;
    absorption_total == 1.0;  -- prevent divide by 0
  end use;

  if domain = quiescent_domain use
    sensor_current == 0.0;
    vout == 0.0;
  else
    sensor_current == (exposure*flux*sensor_area*quantum_efficiency*absorption_ehp*
                        fill_factor*keVp_max*current_keV*q) / (W*absorption_total*time2real(shutterTime));
    i_out == −sensor_current + Cpix*vout ’dot;
  end use;
end medium;

A.2.4 Test Bench

library disciplines, work;
use disciplines.electromagnetic_system.all;
use work.radiant_systems.all;
use work.all;

entity seleniumSensor_bench is
end entity seleniumSensor_bench;

architecture bench_simple of seleniumSensor_bench is
terminal lightSourceTest : radiant;
terminal probe : electrical;

quantity flux_in across dummy through lightSourceTest to radiant_ref;
quantity v_probe across i_probe through probe to electrical_ground;

signal flux_sigh : real := 0.0;
signal enable : bit := '0';

begin
dut : entity seleniumSensor (simple)
generic map(Cst => 1.0e−12,
Cse => 0.01e−12,
conversion_factor => 10000.0)
port map(lightInput => lightSourceTest,
lightRef => radiant_ref,
output => probe,
ref => electrical_ground,
enable => enable);

flux_in == flux_sigh;
flux_sigh <= 0.0, 4.0e4 after 0.5us, 0.0 after 1.0us;
i_probe == 0.0;
break on flux_sigh;

end architecture bench_simple;

architecture bench_medium of seleniumSensor_bench is
terminal lightSourceTest : radiant;
terminal probe : electrical;

quantity flux_in across dummy through lightSourceTest to radiant_ref;
quantity v_probe across i_probe through probe to electrical_ground;

signal flux_sigh : real := 0.0;
signal enable : bit := '0';

begin  --bench_medium
dut : entity seleniumSensor (medium)
generic map(Cst => 1.0e−12,
Cse => 0.01e−12,
keVp_min => 5.0,
keVp_max => 80.0,
shutterTime => 1.0 us,
sensor_area => 0.25*0.25,
W => 0.087,
fill_factor => 0.85)
port map(lightInput => lightSourceTest,
lightRef => radiant_ref,
output => probe,
ref => electrical_ground,
enable => enable);

flux_in == flux_sigh;
flux_sigh <= 0.0, 1.0e5 after 1.0us, 0.0 after 2.0us;
enable <= '0', '1' after 1.0us, '0' after 2.0us;
i_probe == 0.0;
break on flux_sigh;
break on enable;

end architecture bench_medium;

42
A.3 TFT

A.3.1 Entity

library disciplines;
use disciplines.electromagnetic_system.all;

entity tft is
  generic (  
    — for simple model  
    Ron : real := 1.0e6;  
    Roff : real := 1.0e12;  
    Vt : real := 1.0;  

    — for medium model  
    un : real := 0.5;  
    ti : real := 0.3;  
    k : real := 7.0;  
    W : real := 10.0;  
    L : real := 1.0;  
    ta : real := 0.3;  
    n0 : real := 4.3e15;  
    leakage : real := 5.0e-14  
    );
  port (  
    terminal source , drain , gate : electrical;  
    terminal ref : electrical  
  );
end entity tft;

A.3.2 Simple Architecture

architecture simple of tft is
  quantity v_d across i_d through drain to source;  
  quantity v_gate across i_gate through gate to ref;
begin
  i_gate == 0.0;
  if v_gate > Vt use
    i_d == v_d / Ron;
  else
    i_d == v_d / Roff;
  end use;
  break on v_gate'above(Vt);
end architecture simple;

A.3.3 Medium Architecture

architecture medium of tft is
  constant epsilon0 : real := 8.85e-18;  
    — Permittivity of free space (F/um)  
  constant q : real := 1.6e-19;  
    — Electronic charge (C)

  quantity v_d across i_d through drain to source;  
  quantity v_gate across i_gate through gate to ref;  
  quantity Vt_med : real;
  quantity Ci : real := 1.0;
  quantity v_d_sat : real;
begin
  — medium
  — Calculate some simple values
  Ci == k*epsilon0/ti;
  Vt_med == -q*ta*(n0*1.0e-12)/Ci;  
    — Voltage threshold (V)
  v_d_sat == v_gate - Vt_med;

  — Assume no gate leakage
  i_gate == 0.0;

  —--
−−Calculate drain-to-source current
if v_gate > Vt_med use
  if v_d < v_d_sat use
    i_d == un*(Ci*1.0e8)*(W/L)*((v_gate-Vt.med)*v_d-((v_d**2)/2.0)) + leakage;
  else
    i_d == (1.0/2.0)*un*(Ci*1.0e8)*(W/L)*(v_gate - Vt_med)**2 + leakage;
  end use;
else
  i_d == leakage;
end use;

−−These statements work around the discontinuity problem
break on v_gate 'above(Vt_med);
break on v_d 'above(v_d_sat);
end architecture medium;

A.3.4 Test Bench

library disciplines, work.MGC_AMS;
use disciplines.electromagnetic_system.all;
use work.all;
use MGC_AMS.conversion.all;
entity tft_bench is
end entity tft_bench;

architecture vary_gate of tft_bench is
  constant vg_start : real := -20.0; -- Starting gate voltage;
  constant vg_end : real := 20.0; -- End gate voltage
  constant end_time : time := 1 us; -- End time
  constant slope : real := (vg_end - vg_start)/(end_time-1.0); -- Slope of vg ramp
  terminal drain : electrical;
  terminal gate : electrical;
  quantity v_drain across i_drain through drain to electrical.ground;
  quantity v_gate across i_gate through gate to electrical.ground;
begin
  dut : entity tft(medium)
    generic map(un => 0.5,
      ti => 0.3,
      k => 7.0,
      W => 10.0,
      L => 1.0,
      ta => 0.3,
      n0 => 4.3e15,
      leakage => 5.0e-14)
    port map(source => electrical.ground,
      drain => drain,
      gate => gate,
      ref => electrical.ground);
  v_gate == vg_start + slope*now;
  v_drain == 0.1;
end architecture vary_gate;

architecture ids_curves of tft_bench is
  constant vg_start : real := -5.0; -- Starting gate voltage;
  constant vg_end : real := 17.5; -- End gate voltage
  constant vg_steps : real := 6.0;
  constant vg_increment : real := (vg_end - vg_start)/(vg_steps-1.0);
  constant v_d_start : real := 0.0;
  constant v_d_end : real := 20.0;
  constant end_time : time := 1 us; -- End time for one v_d sweep
  terminal drain : electrical;
  quantity v_d across i_drain through drain to electrical.ground;
terminal gate : electrical;

quantity v_dRAIN across i_dRAIN through drain to electrical_ground;
quantity v_GATE across i_GATE through gate to electrical_ground;

signal base_time : time := 0.0 ns;
signal v_GATE_sIG : real := vg_start;

begin
dut : entity tft(medium)
  generic map(un => 0.5,
  tl => 0.3,
  k => 7.0,
  W => 10.0,
  L => 1.0,
  ta => 0.3,
  n0 => 4.3e15,
  leakage => 5.0e-14)
  port map(source => electrical_ground,
  drain => drain,
  gate => gate,
  ref => electrical_ground);

change_gate : process (v_GATE_sIG)
begin
  v_GATE_sIG <= v_GATE_sIG + vg_increment after end_time;
  base_time <= now;
end process change_gate;

v_GATE == v_GATE_sIG;
break on v_GATE_sIG;
  v_dRAIN == ((vd_end - vd_start)/time2real(end_time)) * (now - time2real(base_time));
end architecture ids_curves;
A.4 Amplifier

A.4.1 Entity

library disciplines;
use disciplines.electromagnetic_system.all;

entity amp is

  generic (  
    -- for all models  
    Avol : real := 1.0e15;  
    Cint : real := 5.0e-12;  
    -- for simple model  
    Ron : real := 1.0;  
    Roff : real := 1.0e13;  
    -- for medium model  
    -- for advanced model  
    gain_reduction_factor : real := 0.1;  
    Rpole : real := 1.0e5;  
    Cpole : real := 800.0e-3  
  );

  port (  
    terminal inp : electrical;  
    terminal inm : electrical;  
    terminal outp : electrical;  
    terminal ref : electrical;  
    terminal reset : electrical);  

end entity amp;

A.4.2 Simple Architecture

library work;
use work.all;

architecture simple of amp is

  quantity v_inp_inm across i_inp_inm through inp to inm;
  quantity v_out across i_out through outp to ref;
  quantity v_reset across i_reset through reset to ref;
  quantity Rfeed : real := 1.0e14;
  quantity v_feedback across i_feed through outp to inm;

begin  
  -- simple  
  -- ideal op-amp definition  
  i_inp_inm == 0.0;  
  i_reset == 0.0;  
  
  v_out == Avol * v_inp_inm;
  if v_reset 'above (0.0) use  
    Rfeed == Rom;
  else  
    Rfeed == Roff;
  end use;
  
  v_feedback == i_feedback * Rfeed;

  C_feedback : entity capacitor(simple)  
  generic map(C => Cint,  
    IC => 0.0)  
  port map(p => outp,  
    m => inm);  

end architecture simple;
A.4.3 Medium Architecture

library work;
use work.all;

architecture medium of amp is
  quantity v_inp_inm across i_inp_inm through inp to inm;
  quantity v_out across i_out through outp to ref;
begin
  -- ideal op-amp definition
  i_inp_inm == 0.0;
  v_out == Avol * v_inp_inm;
  reset_switch : entity tft(medium)
    generic map(Ron => 2.0e5, Roff => 2.0e14)
    port map(source => inm, drain => outp, gate => reset, ref => electrical_ground);
C_feedback : entity capacitor(simple)
  generic map(C => Cint, IC => 0.0)
  port map(p => outp, m => inm);
end architecture medium;

A.4.4 Test Bench

library ieee, disciplines, work;
use ieee.math_real.all;
use disciplines.electromagnetic_system.all;
use work.all;

entity amp_bench is
  end entity amp_bench;

architecture bench_simple of amp_bench is
  constant R_load : real := 10.0e3;
  constant freq : real := 1.0e3;
  constant amplitude : real := 1.0;
  constant R_input : real := 2.0e5;
  signal v_reset_sig : real := -5.0;
begin
  -- bench_simple
tut : entity amp(simple)
    generic map(Avol => 1.0e15, Cint => 5.0e-12, Ron => 1.0, Roff => 1.0e13)
    port map(inm => input, inp => electrical_ground, outp => output, ref => electrical_ground, reset => reset);
  rinput : entity resistor
    generic map(R => R_input)
    port map(p => source, m => input);
\[
v_{\text{source}} == \text{amplitude} \times \cos(2\pi \times \text{freq} \times \text{now})\; ;
\]
\[
v_{\text{output}}/R_{\text{load}} == i_{\text{output}}\; ;
\]
\[
v_{\text{reset\_sig}} <= 5.0 \text{ after } 5 \text{ ms}, -5.0 \text{ after } 6.0 \text{ ms};
\]
\[
v_{\text{reset}} == v_{\text{reset\_sig}}\; ;
\]
\[
\text{break on } v_{\text{reset\_sig}}\; ;
\]
end architecture bench\_simple;

architecture bench\_medium of amp\_bench is
constant \( R_{\text{load}} \): real := 1.0e3;
constant \( \text{freq} \): real := 1.0e3;
constant \( \text{amplitude} \): real := 1.0;
constant \( R_{\text{input}} \): real := 2.0e5;
signal \( \text{reset\_sig} \): real := 10.0;
terminal \( \text{source}, \text{input}, \text{output}, \text{reset} \) : electrical;

quantity \( v_{\text{source}} \) across \( i_{\text{source}} \) through \( \text{source} \) to electrical\_ground;
quantity \( v_{\text{output}} \) across \( i_{\text{output}} \) through \( \text{output} \) to electrical\_ground;
quantity \( v_{\text{reset}} \) across \( i_{\text{reset}} \) through \( \text{reset} \) to electrical\_ground;

begin  --- bench\_medium
  dut: entity amp(medium)
  generic map(Avol => 1.0e15, Cint => 5.0e-12)
  port map(inm => input, inp => electrical\_ground, outp => output, ref => electrical\_ground, reset => reset);
\[
v_{\text{source}} == \text{amplitude} \times \cos(2\pi \times \text{freq} \times \text{now})\; ;
\]
  \input : entity resistor
  generic map(R => R_{\text{input}})
  port map(p => source, m => input);
\[
v_{\text{output}}/R_{\text{load}} == i_{\text{output}}\; ;
\]
\[
v_{\text{reset}} == \text{reset\_sig}\; ;
\]
\[
\text{reset\_sig} <= -5.0 \text{ after } 2 \text{ ms}, 10.0 \text{ after } 8 \text{ ms};
\]
\[
\text{break on } \text{reset\_sig}\; ;
\]
end architecture bench\_medium;
A.5  PPS System

A.5.1  Testbench Entity

library  disciplines , work :
use  work . radiant_systems . all :
use  disciplines . electromagnetic_system . all :
use  work . all :

entity  system_bench  is
dend  system_bench :
\end{listing}
\subsection{Simple Testbench}
\begin{listing}
architecture  simple  of  system_bench  is

signal  enableXRaySource  :  bit  :=  ' 0 ' :
signal  v_sensorGate_sig  :  real  :=  0 . 0 :
signal  v_ampGate_sig  :  real  :=  0 . 0 :
signal  sensorEnable  :  bit  :=  ' 1 ' :

terminal  lightJunction  :  radiant :
terminal  sensorOutput  :  electrical :
terminal  sensorGate  :  electrical :
terminal  ampInput  :  electrical :
terminal  ampGate  :  electrical :
terminal  output  :  electrical :

quantity  v_output  across  i_output  through  output  to  electrical_ground :
quantity  v_sensorGate  across  i_sensorGate  through  sensorGate  to  electrical_ground :
quantity  v_ampGate  across  i_ampGate  through  ampGate  to  electrical_ground :

begin
−−  simple
myXRaySource  :  entity  xraySource ( simple )
generic  map ( conversionFactor  =>  2 . 5 e 5 ,
keVp_max  =>  80 . 0 )
port  map ( enable  =>  enableXRaySource ,
lightOutput  =>  lightJunction ,
ref_radiant  =>  radiant_ref );

mySensor  :  entity  seleniumSensor ( simple )
generic  map ( Cst  =>  1 . 0 e − 12 ,
Cse  =>  0 . 01 e − 12 ,
conversion_factor  =>  10000 . 0 )
port  map ( lightInput  =>  lightJunction ,
litRef  =>  radiant_ref ,
output  =>  sensorOutput ,
ref  =>  electrical_ground ,
enable  =>  sensorEnable );

myTFT  :  entity  tft ( simple )
generic  map ( Ron  =>  1 . 0 e 6 ,
Roff  =>  1 . 0 e 12 ,
Vt  =>  − 1 . 0 )
port  map ( source  =>  ampInput ,
drain  =>  ampGate ,
gate  =>  sensorOutput ,
ref  =>  electrical_ground );

myAmp  :  entity  amp ( simple )
generic  map ( Avol  =>  1 . 0 e 15 ,
Cint  =>  5 . 0 e − 12 ,
Ron  =>  1 . 0 ,
Roff  =>  1 . 0 e 13 )
port  map ( inp  =>  electrical_ground ,
inm  =>  ampInput ,
outp  =>  output ,
}
David James Grant

X-Ray pixel model in VHDL-AMS

ref => electrical_ground,
reset => ampGate);

enableXRaySource <= '0', '1' after 1 us, '0' after 4 us;

v_sensorGate_sig <= -5.0, 10.0 after 5 us;
v_sensorGate == v_sensorGate_sig;

v_ampGate_sig <= 10.0, -10.0 after 5 us;
v_ampGate == v_ampGate_sig;

v_output/1.0e3 == i_output;
break on v_sensorGate_sig;
break on v_ampGate_sig;
end architecture simple;

A.5.2 Medium Testbench

architecture medium of system_bench is

signal enableXRaySource : bit := '0';
signal v_sensorGate_sig : real := 0.0;
signal v_ampGate_sig : real := 0.0;
signal sensorEnable : bit := '0';

terminal lightJunction : radiant;
terminal sensorOutput : electrical;
terminal sensorGate : electrical;
terminal ampInput : electrical;
terminal ampGate : electrical;
terminal output : electrical;

quantity v_output across i_output through output to electrical_ground;
quantity v_sensorGate across i_sensorGate through sensorGate to electrical_ground;
quantity v_ampGate across i_ampGate through ampGate to electrical_ground;
begin — medium

myXRaySource : entity xraySource (medium)
generic map(shutterTime => 3 us,
keVp_max => 80.0,
keVp_min => 5.0,
exposure_matlab => 1.249)
port map(enable => enableXRaySource,
lightOutput => lightJunction,
ref_radiant => radiant_ref);

mySensor : entity seleniumSensor (medium)
generic map(Cst => 1.0e-12,
Cse => 0.1e-12,
shutterTime => 3 us,
keVp_max => 80.0,
keVp_min => 5.0,
sensor_area => 0.25*0.25,
W => 0.087,
fill_factor => 0.85)
port map(lightInput => lightJunction,
lightRef => radiant_ref,
output => sensorOutput,
ref => electrical_ground,
enable => sensorEnable);

myTFT : entity tft (medium)
port map(source => ampInput,

drain => sensorOutput,

gate => sensorGate,

ref => electrical_ground);
myAmp : entity amp(medium)
    port map(inp => electrical_ground,
              inm => ampInput,
              outp => output,
              ref => electrical_ground,
              reset => ampGate);

enableXRaySource <= '0', '1' after 1 us, '0' after 4 us;
sensorEnable <= '0', '1' after 1 us, '0' after 4 us;

v_sensorGate_sig <= -5.0, 10.0 after 5 us;
v_sensorGate == v_sensorGate_sig;

v_ampGate_sig <= 10.0, -10.0 after 5 us;
v_ampGate == v_ampGate_sig;

v_output/1.0e3 == i_output;

break on v_sensorGate_sig;
break on v_ampGate_sig;
end architecture medium;
A.6 APS System

A.6.1 Testbench Entity

library disciplines, work;
use work.radiant_systems.all;
use disciplines.electromagnetic_system.all;
use work.all;

entity system_bench is
end system_bench;

A.6.2 Testbench

architecture active of system_bench is
  signal enableXRaySource : bit := '0';
  signal v_readGate_sig : real := 0.0;
  signal v_ampGate_sig : real := 0.0;
  signal v_resetGate_sig : real := 0.0;
  signal sensorEnable : bit := '0';

terminal lightJunction : radiant;
terminal sensorOutput : electrical;
terminal readGate : electrical;
terminal ampInput : electrical;
terminal ampGate : electrical;
terminal output : electrical;
terminal resetGate : electrical;
terminal power : electrical;
terminal commonNode : electrical;

quantity v_output across i_output through output to electrical_ground;
quantity v_readGate across i_readGate through readGate to electrical_ground;
quantity v_ampGate across i_ampGate through ampGate to electrical_ground;
quantity v_resetGate across i_resetGate through resetGate to electrical_ground;
quantity v_power across i_power through power to electrical_ground;

begin -- active
myXRaySource : entity xraySource(medium)
generic map(shutterTime => 2 us,
keVp_max => 80.0,
keVp_min => 5.0,
exposure_matlab => 1.249)
port map(enable => enableXRaySource,
lightOutput => lightJunction,
ref_radian => radiant_ref);

mySensor : entity seleniumSensor(medium)
generic map(Cst => 0.5e-12,
Cse => 0.01e-12,
shutterTime => 2 us,
keVp_max => 80.0,
keVp_min => 5.0,
sensor_area => 0.25*0.25,
W => 0.087,
fill_factor => 0.85)
port map(lightInput => lightJunction,
lightRef => radiant_ref,
output => electrical_ground,
ref => sensorOutput,
enable => sensorEnable);

resetTFT : entity tft(medium)
port map(source => sensorOutput,
drain => power,
gate => resetGate,
ref => electrical_ground);
ampTFT : entity tft (medium)
    port map (source => commonNode,
               drain => power,
               gate => sensorOutput,
               ref => power);

readOutTFT : entity tft (medium)
    port map (source => ampInput,
              drain => commonNode,
              gate => readGate,
              ref => electricalGround);

myAmp : entity amp (simple)
    port map (inp => electricalGround,
              inm => ampInput,
              outp => output,
              ref => electricalGround,
              reset => ampGate);

enableXRaySource <= '0', '1' after 10 us, '0' after 12 us;
sensorEnable <= '0', '1' after 10 us, '0' after 12 us;

v_readGate_sig <= -5.0, 10.0 after 13 us, -5.0 after 15 us,
                 10.0 after 17us, -5.0 after 19 us;

v_readGate == v_readGate_sig;

v_ampGate_sig <= 10.0, -10.0 after 13 us, 10.0 after 16 us, -10.0 after 17 us;

v_ampGate == v_ampGate_sig;

v_resetGate_sig <= 10.0, -10.0 after 9 us, 10.0 after 13 us;

v_resetGate == v_resetGate_sig;

v_output/1.0e3 == i_output;

v_power == 10.0;

break on v_readGate_sig;
break on v_ampGate_sig;
break on v_resetGate_sig;
end architecture active;
A.7 Other

A.7.1 Resistor

library disciplines;
use disciplines.electromagnetic_system.all;

entity resistor is
  generic(
    R : real := 1.0);
  port(
    terminal p,m : electrical);
end entity resistor;

architecture impedance of resistor is
  quantity v across i through p to m;
begin
  v == R * i;
end architecture impedance;

A.7.2 Capacitor

library disciplines;
use disciplines.electromagnetic_system.all;

entity capacitor is
  generic(
    C : real := 1.0;
    IC : real := 0.0);
  port(
    terminal p,m : electrical);
end entity capacitor;

architecture simple of capacitor is
  quantity v across i through p to m;
begin
  if domain = quiescent_domain use
    v == IC;
  else
    v == IC when IC /= real 'low;
    i == C*v'dot;
  end use;
end architecture simple;

A.7.3 Light Nature

package RADIANT_SYSTEMS is
  subtype DUMMY is REAL tolerance "DEFAULT_DUMMY";
  subtype OPTIC_FLUX is REAL tolerance "DEFAULT_OPTIC_FLUX";
  -- nature declarations
  nature RADIANT is
    OPTIC_FLUX across
    DUMMY through
    RADIANT_REF reference;
end package RADIANT_SYSTEMS;
References


