To design a laser system with a beam splitter and a photodetector in MATLAB or ANSYS, you need to break down the key elements and the physical equations that govern them. Below I give you a conceptual description and an approach to coding the system in MATLAB.

Main components:

1. **Laser:**
    - It can be modeled using coherent light emission equations.
    - The basic equation for laser power could be represented as: Plaser=P0e−αLP_{\text{laser}} = P_0 e^{- \alpha L}Plaser=P0e−αL where P0P_0P0 is the initial power, α\alphaα is the absorption coefficient, and L is the length of the active medium.
2. **Beam Splitter:**
    - This device splits the laser beam into two paths. Its behavior can be modeled using a matrix of transmission and reflection coefficients.
    - The transmission and reflection relationship could be expressed in terms of the intensities:
    - Itransmitted=T⋅Iincident,Ireflected=R⋅IincidentI_{\text{transmitted}} = T \cdot I_{\text{incident}}, \quad I_{\text{reflected}} = R \cdot I_{\text{ incident}}Itransmitted=T⋅Iincident,Ireflected=R⋅Iincident where TTT is the transmission coefficient and RRR is the reflection coefficient, satisfying T+R=1T + R = 1T+R=1.
3. **Fotodetector:**
    - Converts optical energy into an electrical signal. It can be modeled using the current generated as a function of the power of the incident beam: Iphotodetector=ηPincidenteI_{\text{photodetector}} = \eta P_{\text{incident}}Ifotodetector=ηPincidente where η\etaη is the quantum efficiency of the photodetector.

Code in MATLAB:

The following example illustrates how to model a laser, a beam splitter and a photodetector:

```
% Initial parameters
P0 = 1; % Laser power (W)
alpha = 0.1; % Absorption coefficient
L = 10; % Length of active medium (m)

% Beam Splitter
T = 0.5; % Transmission coefficient
R = 1 - T; % Reflection coefficient
```

```
% Power after laser
P_laser = P0 * exp(-alpha * L);

% Intensities after beam splitter
P_transmitted = T * P_laser;
P_reflected = R * P_laser;

% Fotodetector
eta = 0.9; % Quantum efficiency of the photodetector
I_photodetector_transmitted = eta * P_transmitted;
I_photodetector_reflected = eta * P_reflected;

% Results
fprintf('Power transmitted: %.4f W\n', P_transmitted);
fprintf('Reflected power: %.4f W\n', P_reflected);
fprintf('Current in photodetector (transmitted): %.4f A\n',
I_photodetector_transmitted);
fprintf('Current in photodetector (reflected): %.4f A\n',
I_photodetector_reflected);
```

Circuit design in ANSYS:

For ANSYS, you could use optical simulation to model the propagation of the laser beam through a beam splitter and photodetectors. ANSYS supports simulations based on beam propagation theory, and you could use its Optical Systems Design module to create the components:

1. **Laser:**
    - Define the laser source with wavelength and power.
2. **Beam Splitter:**
    - Use a split surface object with adjustable transmission and reflection coefficients.
3. **Fotodetector:**
    - Define a detector in the simulation that captures the split beam and converts the optical energy into an electrical signal.

Code and Libraries in ANSYS (Lumerical)

In ANSYS Lumerical, you can use the software's own language (Lumerical Script) or Python to automate optical simulations. Here I give you a basic example of how you could simulate a laser.

1. Environment configuration

First, the simulation is defined with the laser parameters, such as the wavelength and the geometry of the laser cavity.

2. Definition of the light source (laser)

You can create a coherent laser source in the time domain or frequency domain.

3. Code in Lumerical Script

Here is an example of how a simple laser can be configured in a Lumerical FDTD (Finite-Difference Time-Domain) environment:

```
# Create a new project in Lumerical FDTD
newproject;

# Define the wavelength of the laser (in microns)
lambda0 = 1.55e-6;  # Wavelength in meters (1.55 microns)

# Add a laser source in the time domain (Gaussian)
addfdtd;
select("FDTD");

# Define the dimensions of the simulator
set("x span", 10e-6);  # Width in X direction
set("and span", 10e-6);  # Width in Y direction
set("z span", 5e-6);   # Width in Z direction
set("wavelength start", lambda0 - 0.1e-6);  # Initial wavelength
range
set("wavelength stop", lambda0 + 0.1e-6);   # Final wavelength
range

# Add a coherent light source (laser)
addsource;  # Add source
set("name", "laser_source");  # Name the source
set("injection axis", "z");  # Injection direction
set("wavelength", lambda0);  #Laser wavelength

# Laser settings
set("pulse type", "continuous wave");  # Continuous wave
set("power", 1);  # Output power (W)

# Add monitor to capture beam intensity
addpower;
```

```
set("monitor type", "2D X-normal");  # Monitor de campo 2D
set("x", 0);  # Monitor position in X

# Run the simulation
run;
```

Code description:

1. **newproject**: Create a new project for the simulation.
2. **addfdtd**: Defines the simulation environment using the FDTD method.
3. **addsource**: Adds a laser source with a specific wavelength. You can adjust parameters such as injection direction, pulse type (continuous or modulated), and power.
4. **addpower**: This block adds a power monitor to measure the beam intensity in the simulation.
5. **run**: Run the simulation and calculate the propagated electric and magnetic fields.

4. Viewing Results

Once the simulation is run, the results can be visualized using Lumerical's graphical interface, where graphs of the field distribution and laser intensity are generated.

```
import lumapi

# Create an instance of the FDTD simulator
fdtd = lumapi.FDTD()

# Define laser wavelength
lambda0 = 1.55e-6 # 1.55 micrometers

# Configure the simulator
fdtd.addfdtd()
fdtd.set("x span", 10e-6)
fdtd.set("y span", 10e-6)
fdtd.set("z span", 5e-6)
fdtd.set("wavelength start", lambda0 - 0.1e-6)
fdtd.set("wavelength stop", lambda0 + 0.1e-6)

# Add a laser source
fdtd.addsource()
fdtd.set("name", "laser_source")
fdtd.set("injection axis", "z")
fdtd.set("wavelength", lambda0)
fdtd.set("pulse type", "continuous wave")
fdtd.set("power", 1)

# Add power monitor
fdtd.addpower()
```

```
fdtd.set("monitor type", "2D X-normal")
fdtd.set("x", 0)

# Run the simulation
fdtd.run()

# Extract and view results
results = fdtd.getresult('laser_source', 'field')
print(results)
```

Code description:

- **come closer**: This is the Python library that interacts with the ANSYS Lumerical environment.
- **FDTD**: An instance of the finite difference simulator is created in the time domain.
- **addsource()**: Add a laser source with custom features.
- **run()**: Run the simulation, and the method `getresult()` retrieves the results.

Considerations:

1. **Laser Power and Phase Control:** You can adjust the laser power and control the phase of the wave to perform more complex simulations.
2. **Beam Propagation:** In addition to laser design, you can simulate the propagation of the laser beam through various materials and analyze how it interacts with mirrors, beam splitters, and other optical components.
3. **Optoelectronic Components:** If you are also interested in including devices such as photodetectors or modulators, ANSYS Lumerical offers specific modules to analyze the interaction between light and matter.

In **COMSOL Multiphysics**, you can design a laser system **LIDAR** and connect it with **MATLAB** using the optics module, which allows simulating the interaction of the laser beam with the environment, propagation, scattering and the detection of objects at a distance.

Design of a LIDAR with COMSOL and MATLAB

The LIDAR (Light Detection and Ranging) system uses laser pulses to measure distances based on the time it takes for light to be reflected from an object. Below I will provide you with a

step-by-step guide to designing a simple LIDAR system in **COMSOL** and control it by **MATLAB**.

1. Environment configuration in COMSOL:

- We will use the **Ray Optics Module** from COMSOL, which allows simulating the propagation of rays in optical systems and measuring how they interact with surfaces and media.

2. Define the lidar parameters:

Important variables that you can define for a LIDAR laser:

- **laser wavelength**: Must be in the near infrared range (e.g. 905nm or 1550nm).
- **Pulse repetition rate**: Controls how many pulses per second are sent.
- **Scan angle**: The field of vision covered by the laser.
- **Maximum distance**: Maximum distance at which the laser can detect objects.
- **Time of Flight (ToF)**: The time it takes for light to travel to the object and return to the sensor.

3. Design in COMSOL:

In COMSOL, you can use the MATLAB interface to set up simulations and control variables. Below I give you example code to create a LIDAR model using MATLAB and the COMSOL API.

4. MATLAB code with COMSOL API:

```
% Create model in COMSOL
import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

% Create geometry
geom1 = model.geom.create('geom1', 3);
model.geom('geom1').lengthUnit('m');

% Add an object that the LIDAR will detect
block = geom1.feature.create('blk1', 'Block');
block.set('size', {'0.2' '0.2' '0.2'}); % Object size
block.set('pos', {'1.0' '0.0' '0.0'});  % Object position

% Create a laser (light source)
optics = model.physics.create('rte', 'RayOptics', 'geom1');
laser = optics.feature.create('laser1', 'ReleaseFromGrid');
```

```
laser.set('wavelength', 905e-9);  % LIDAR laser wavelength in meters
(905 nm)
laser.set('numrays', 100);        % Number of rays to emit
laser.set('power', 1);            % Laser power (W)

% Define environmental conditions
optics.feature('laser1').set('material', 'Air');

% Add detector (reflective surface)
detector = geom1.feature.create('det', 'Block');
detector.set('size', {'0.1' '0.1' '0.01'}); % Detector dimensions
detector.set('pos', {'1.0' '0.0' '0.0'});  % Detector position

% Angular Scan Settings
model.study.create('std1');
model.study('std1').feature.create('time', 'Transient');
model.study('std1').feature('time').set('tlist', 'range(0, 1e-9,
1e-6)');  % Simulation time

% Run the simulation
model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').runAll;

% Get distance and flight time results
results = model.result.create('pg1', 'PlotGroup3D');
rayResults = results.create('ray1', 'Ray');
rayResults.feature.create('ref1', 'Reflectance');

% Show results
model.result('pg1').run;
```

Code explanation:

1. **Import and create the model**: You import the COMSOL classes and create the 3D model where the simulation will be developed.
2. **Geometry**: You define the environment in which the laser will propagate. In this case, there is a block that simulates the object that the LIDAR must detect.
3. **laser source**: A laser is created with a wavelength of 905 nm, which is common in LIDAR systems. The number of rays and power are also specified.
4. **Environmental conditions**: The propagation of the laser in air is simulated.
5. **Detector**: The detector is placed in the same position as the object to record the laser reflection.
6. **Transient simulation**: It is specified that the simulation is a function of time (transient), which is important to measure the time of flight of the laser.
7. **Results**: A display of the reflected rays is generated and distances are calculated based on flight time.

Effectiveness Adjustments:

To improve the effectiveness of the LIDAR system, you can adjust several parameters in the code:

- **Scan angle**: You can add a loop that changes the laser emission angle, simulating a sweep in 3D space.
- **Noise and precision**: To simulate real conditions, you can add a noise component to the distance data to evaluate how the system responds to disturbances.
- **Pulse rate**: Increase the repetition rate to obtain more data per second and improve resolution.

COMSOL-MATLAB connection:

- **Advanced LIDAR**: If you want to integrate real-time data processing, you can extend the code in MATLAB to process the data obtained in COMSOL, such as doing signal filtering, identifying objects, etc.
- **Simulation of dynamic scenarios**: You can simulate moving objects and calculate relative speed using the change in flight time.

Effectiveness Variables:

1. **System resolution**: Depends on the pulse frequency and the number of rays emitted.
2. **Maximum range**: It is determined by the power of the laser and the ability of the detector to collect signals over long distances.
3. **Distance Accuracy**: Noise in the environment can affect flight time, so you can add noise simulations to see how accuracy varies.

This is a basic layout that you can expand. If you have more details you would like to include, such as material types or different geometries