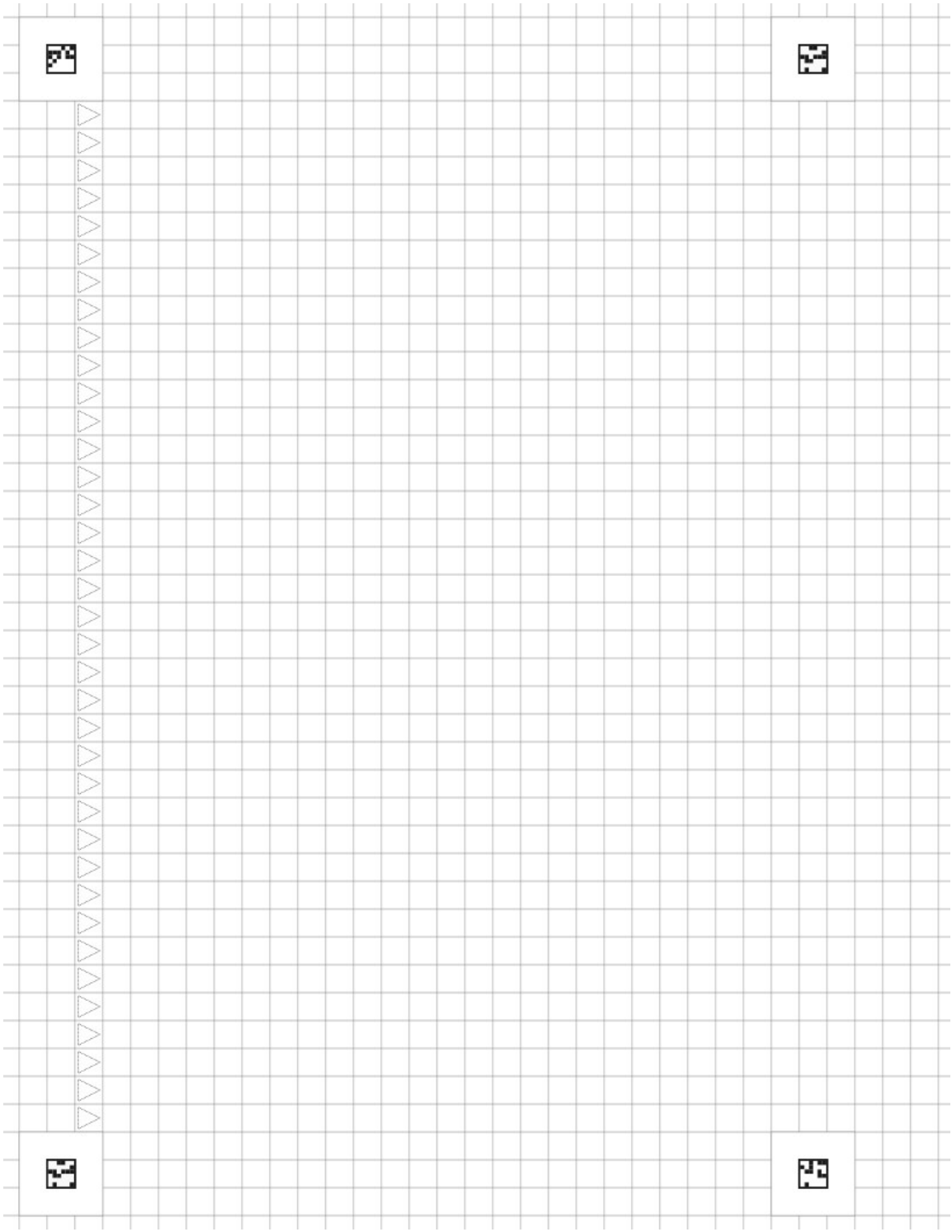
A3 Page Types and Features

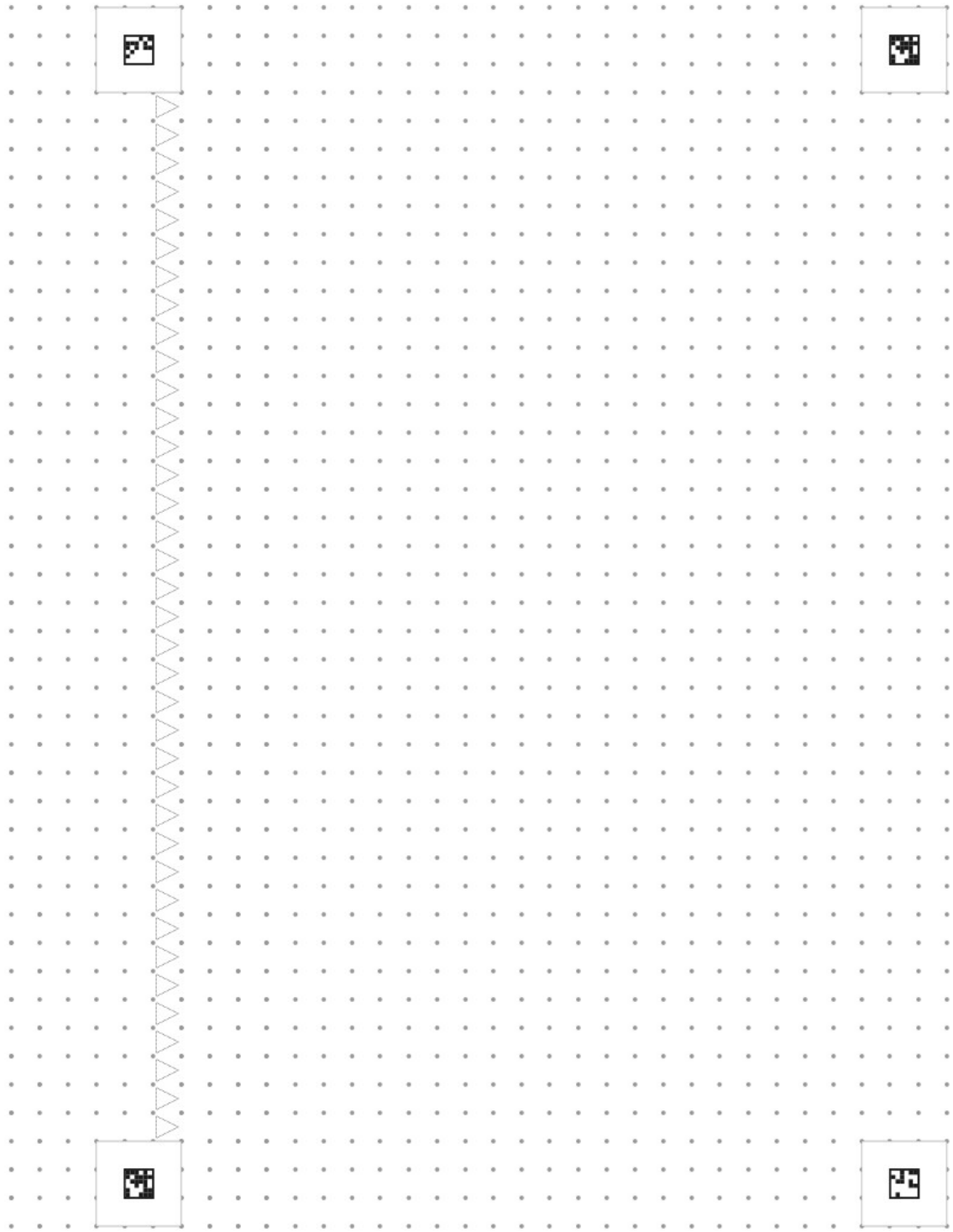
I am working on different kinds of pages for my personal applications. I am also working on different features to help with administrative work.



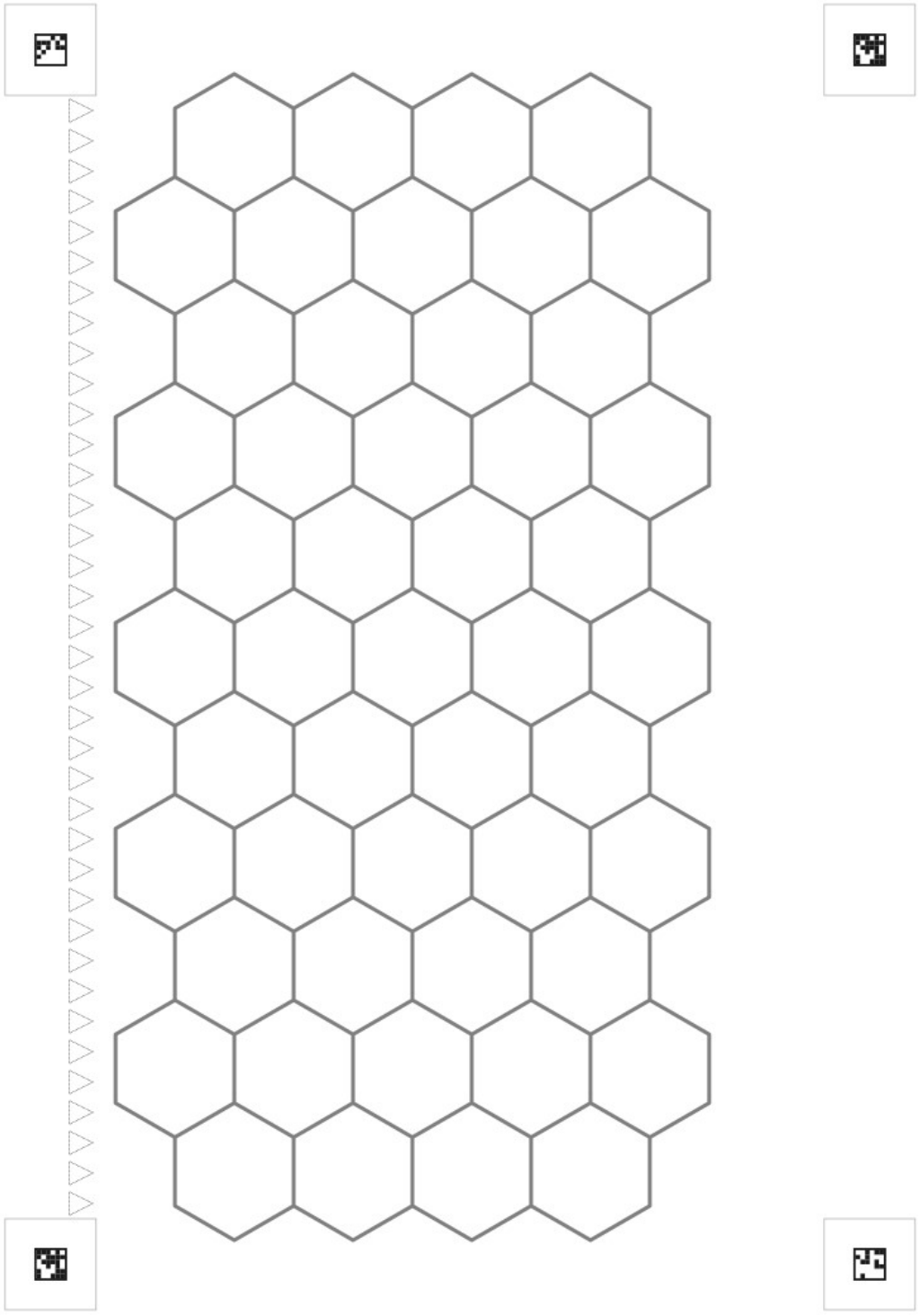
Regular Page

Lined Page

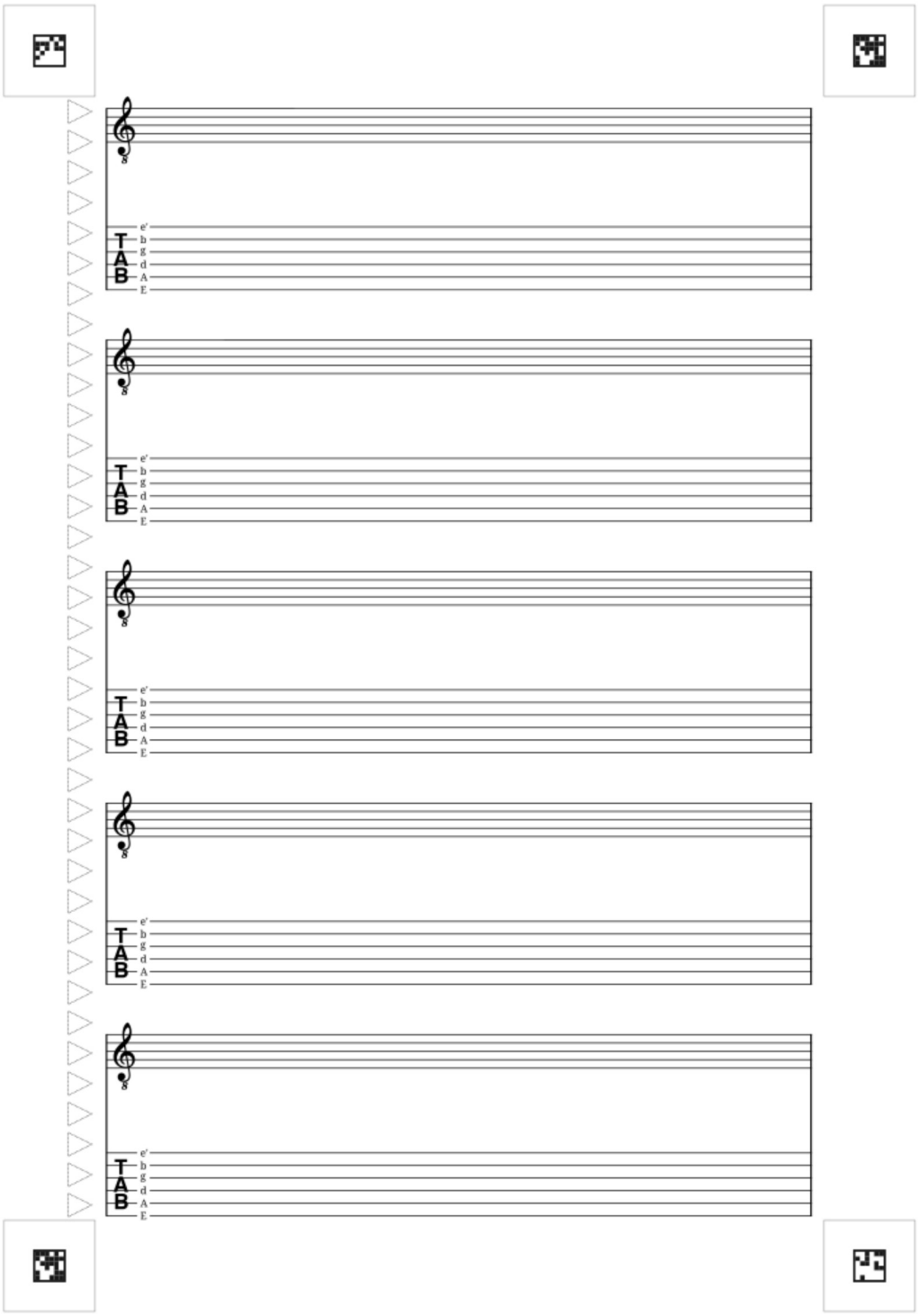
Dot Matrix



Hex



Musical Staff and Guitar Tab



Anti-Cheat Page Signer Code

Here is my `homework_signer.py` code that is meant to sign a `.pdf` to look like the example in **A4**

In [ ]:
```
try:
    import svgwrite
    HAS_SVGWRITE = True
except ImportError:
```

```
    HAS_SVGWRITE = False

import xml.etree.ElementTree as ET
from xml.dom import minidom
import csv
import os
import argparse
import sys
import base64

# PDF processing imports
try:
    import fitz  # PyMuPDF
    HAS_PYMUPDF = True
except ImportError:
    HAS_PYMUPDF = False

# 4-state barcode generator for Royal Mail (RM4SCC-like)
# Following basic RM4SCC (often used for postal barcodes): F, A, D, T (tracker, asc

# Map each character to barcode bars (12 characters max, can be extended)
CHAR_MAP = {
    'A': 'ATDA', 'B': 'ADTA', 'C': 'AATD', 'D': 'ADAT', 'E': 'TADA', 'F': 'TDAA',
    'G': 'TDDA', 'H': 'TADA', 'I': 'DTAA', 'J': 'DATD', 'K': 'DAAT', 'L': 'DAAD',
    'M': 'ATAD', 'N': 'TDAA', 'O': 'TDAD', 'P': 'TDDA', 'Q': 'ATAA', 'R': 'AADT',
    'S': 'AADT', 'T': 'AATA', 'U': 'ATDD', 'V': 'TADD', 'W': 'TDDA', 'X': 'TADA',
    'Y': 'DTAA', 'Z': 'DATD',
    '0': 'ADDA', '1': 'DADA', '2': 'DAAD', '3': 'ADAD', '4': 'DADA', '5': 'DAAD',
    '6': 'DDAA', '7': 'DADA', '8': 'DAAD', '9': 'DDAA',
    # Add mappings as needed
}

def char_to_bars(c):
    c = c.upper()
    return CHAR_MAP.get(c, 'ATDA')  # default fallback

def encode_4state_barcode(data):
    bars = []
    for c in data:
        bars.extend(char_to_bars(c))
    return bars

def generate_4state_barcode_svg(data, filename='barcode.svg', bar_width=4, space=2,
    if not HAS_SVGWRITE:
        raise ImportError("svgwrite is required for generate_4state_barcode_svg. In
    bars = encode_4state_barcode(data)
    dwg = svgwrite.Drawing(filename, size=(len(bars) * (bar_width + space), height_
    x = 0
    y_full = 0
    y_tracker = (height_full - tracker_height) // 2
    for bar in bars:
        if bar == 'F':  # Full
            dwg.add(dwg.rect((x, y_full), (bar_width, height_full), fill='black'))
        elif bar == 'A':  # Ascender
            dwg.add(dwg.rect((x, y_full), (bar_width, height_asc), fill='black'))
        elif bar == 'D':  # Descender
```

```
            dwg.add(dwg.rect((x, height_full - height_desc), (bar_width, height_des
        elif bar == 'T':  # Tracker
            dwg.add(dwg.rect((x, y_tracker), (bar_width, tracker_height), fill='bla
        x += bar_width + space
    dwg.save()

def insert_barcode_into_svg(input_svg_path, barcode_data, output_svg_path=None, bar
    """
    Insert a barcode into an existing SVG document at the rect element with ID 'aus
    If no auspost rect is found, adds barcode at specified position or default bott

    Args:
        input_svg_path: Path to the input SVG file
        barcode_data: String data to encode as barcode
        output_svg_path: Path to save the output SVG (defaults to input_svg_path)
        bar_width: Width of each bar (auto-calculated if None)
        space: Space between bars (auto-calculated if None)
        label_text: Text to display in the label (defaults to barcode_data if None)
        position: Tuple (x, y) for barcode position if no auspost rect found (defau
    """
    if output_svg_path is None:
        output_svg_path = input_svg_path

    # Parse the existing SVG
    tree = ET.parse(input_svg_path)
    root = tree.getroot()

    # Register namespaces to handle SVG properly
    namespaces = {'svg': 'http://www.w3.org/2000/svg'}
    svg_namespace = None
    if root.tag.startswith('{'):
        # Extract namespace from root tag
        ns = root.tag.split('}')[0].strip('{')
        namespaces[''] = ns
        svg_namespace = ns
    else:
        # Check for default namespace in xmlns attribute
        if 'xmlns' in root.attrib:
            svg_namespace = root.attrib['xmlns']
        else:
            svg_namespace = 'http://www.w3.org/2000/svg'

    # Find ALL rects with ID "auspost" or inkscape:label="auspost" (one per page)
    auspost_rects = []
    auspost_label_texts = []

    def find_auspost_recursive(elem, parent=None):
        nonlocal auspost_rects, auspost_label_texts
        # Handle both namespaced and non-namespaced elements
        tag = elem.tag.split('}')[-1] if '}' in elem.tag else elem.tag
        if tag == 'rect':
            # Check for id="auspost" or inkscape:label="auspost"
            elem_id = elem.get('id', '')
            # Check for inkscape:label attribute (may be namespaced)
            inkscape_label = None
            for attr_name, attr_value in elem.attrib.items():
```

```
                if attr_name.endswith('label') and attr_value == 'auspost':
                    inkscape_label = attr_value
                    break
            if elem_id == 'auspost' or inkscape_label == 'auspost':
                auspost_rects.append((elem, parent))
        elif tag == 'text':
            # Check for inkscape:label="auspost_label"
            inkscape_label = None
            for attr_name, attr_value in elem.attrib.items():
                if attr_name.endswith('label') and attr_value == 'auspost_label':
                    inkscape_label = attr_value
                    break
            if inkscape_label == 'auspost_label':
                auspost_label_texts.append(elem)
        # Recursively search children
        for child in elem:
            find_auspost_recursive(child, elem)

    find_auspost_recursive(root)

    # Get SVG dimensions for fallback positioning
    # Try viewBox first (most reliable)
    if 'viewBox' in root.attrib:
        viewbox_parts = root.attrib['viewBox'].split()
        if len(viewbox_parts) >= 4:
            svg_width = float(viewbox_parts[2])
            svg_height = float(viewbox_parts[3])
        else:
            # Fallback to width/height attributes
            width_attr = root.get('width', '612')
            height_attr = root.get('height', '792')
            # Handle units like '8.5in'
            if 'in' in str(width_attr):
                svg_width = float(str(width_attr).replace('in', '').strip()) * 96
            else:
                svg_width = float(width_attr)
            if 'in' in str(height_attr):
                svg_height = float(str(height_attr).replace('in', '').strip()) * 96
            else:
                svg_height = float(height_attr)
    else:
        # Parse width/height attributes with unit handling
        width_attr = root.get('width', '612')
        height_attr = root.get('height', '792')
        # Handle units like '8.5in'
        if 'in' in str(width_attr):
            svg_width = float(str(width_attr).replace('in', '').strip()) * 96
        else:
            svg_width = float(width_attr)
        if 'in' in str(height_attr):
            svg_height = float(str(height_attr).replace('in', '').strip()) * 96
        else:
            svg_height = float(height_attr)

    # Encode the barcode
    bars = encode_4state_barcode(barcode_data)
```

```
    num_bars = len(bars)

    # If no auspost rects found, use default position
    if not auspost_rects:
        # Default to bottom-right corner with padding
        if position is None:
            # Calculate default size for barcode area
            default_width = min(200, svg_width * 0.3)
            default_height = 50
            default_x = svg_width - default_width - 20  # 20px padding from right
            default_y = svg_height - default_height - 20  # 20px padding from botto
            position = (default_x, default_y)
            rect_width = default_width
            rect_height = default_height
        else:
            default_width = min(200, svg_width * 0.3)
            default_height = 50
            rect_width = default_width
            rect_height = default_height
    else:
        # Use the first rect for dimensions (they should all be the same size)
        first_rect, _ = auspost_rects[0]
        rect_width = float(first_rect.get('width', 100))
        rect_height = float(first_rect.get('height', 50))

    # Calculate bar dimensions to fit within the rect
    if bar_width is None or space is None:
        # Auto-calculate to fit the width
        # Total width needed: num_bars * bar_width + (num_bars - 1) * space
        # We want this to fit in rect_width with some padding
        available_width = rect_width * 0.95  # 95% of width for padding
        if num_bars > 1:
            # bar_width = available_width / (num_bars + (num_bars - 1) * space_rati
            # Assuming space = bar_width / 2 for good appearance
            space_ratio = 0.5
            bar_width = available_width / (num_bars * (1 + space_ratio) - space_rat
            space = bar_width * space_ratio
        else:
            bar_width = available_width
            space = 0

    # Calculate heights to fit within rect_height
    height_full = rect_height * 0.9  # 90% of height
    height_asc = height_full * 0.8
    height_desc = height_full * 0.8
    tracker_height = height_full * 0.2

    # Calculate total barcode width for centering
    total_barcode_width = num_bars * bar_width + (num_bars - 1) * space

    # Remove the original rects or make them invisible (for all pages)
    if auspost_rects:
        for auspost_rect, _ in auspost_rects:
            auspost_rect.set('fill', 'none')
            auspost_rect.set('stroke', 'none')
```

```
    # Find all page layers (page 0 and page 1)
    page_layers = []
    for elem in root.iter():
        tag = elem.tag.split('}')[-1] if '}' in elem.tag else elem.tag
        if tag == 'g':
            # Check for inkscape:label starting with "page" or id="layer2"/"layer3"
            elem_id = elem.get('id', '')
            inkscape_label = None
            for attr_name, attr_value in elem.attrib.items():
                if attr_name.endswith('label') and attr_value and attr_value.starts
                    inkscape_label = attr_value
                    break
            if elem_id in ['layer2', 'layer3'] or (inkscape_label and inkscape_labe
                page_layers.append(elem)

    # Determine where to insert barcode
    if auspost_rects:
        # Match auspost rects with page layers
        # Insert barcode rects into each page using the corresponding auspost rect
        for i, (auspost_rect, auspost_parent) in enumerate(auspost_rects):
            # Get the rect position for this specific page
            rect_x = float(auspost_rect.get('x', 0))
            rect_y = float(auspost_rect.get('y', 0))

            # Calculate starting position (centered horizontally) for this page
            start_x = rect_x + (rect_width - total_barcode_width) / 2
            start_y = rect_y + (rect_height - height_full) / 2
            y_tracker = start_y + (height_full - tracker_height) / 2

            # Find the corresponding page layer parent
            if page_layers and i < len(page_layers):
                parent = page_layers[i]
            else:
                parent = auspost_parent if auspost_parent is not None else root

            # Insert barcode rects for this page
            x_pos = start_x
            for bar in bars:
                # Create rect element with proper namespace
                if svg_namespace:
                    rect_elem = ET.Element('{' + svg_namespace + '}rect')
                else:
                    rect_elem = ET.Element('rect')
                # Set fill color as requested - using style attribute for better co
                rect_elem.set('style', 'fill:#CCCCCCFF;fill-opacity:1')
                # Also set as attribute for maximum compatibility
                rect_elem.set('fill', '#CCCCCCFF')

                if bar == 'F':  # Full
                    rect_elem.set('x', str(x_pos))
                    rect_elem.set('y', str(start_y))
                    rect_elem.set('width', str(bar_width))
                    rect_elem.set('height', str(height_full))
                elif bar == 'A':  # Ascender
                    rect_elem.set('x', str(x_pos))
                    rect_elem.set('y', str(start_y))
```

```
                    rect_elem.set('width', str(bar_width))
                    rect_elem.set('height', str(height_asc))
                elif bar == 'D':  # Descender
                    rect_elem.set('x', str(x_pos))
                    rect_elem.set('y', str(start_y + height_full - height_desc))
                    rect_elem.set('width', str(bar_width))
                    rect_elem.set('height', str(height_desc))
                elif bar == 'T':  # Tracker
                    rect_elem.set('x', str(x_pos))
                    rect_elem.set('y', str(y_tracker))
                    rect_elem.set('width', str(bar_width))
                    rect_elem.set('height', str(tracker_height))

                parent.append(rect_elem)
                x_pos += bar_width + space
    else:
        # No auspost rects found - add barcode at specified or default position
        if position is None:
            position = (svg_width - rect_width - 20, svg_height - rect_height - 20)

        start_x = position[0] + (rect_width - total_barcode_width) / 2
        start_y = position[1] + (rect_height - height_full) / 2
        y_tracker = start_y + (height_full - tracker_height) / 2

        # Insert barcode rects directly into root
        x_pos = start_x
        for bar in bars:
            # Create rect element with proper namespace
            if svg_namespace:
                rect_elem = ET.Element('{' + svg_namespace + '}rect')
            else:
                rect_elem = ET.Element('rect')
            # Set fill color as requested - using style attribute for better compat
            rect_elem.set('style', 'fill:#CCCCCCFF;fill-opacity:1')
            # Also set as attribute for maximum compatibility
            rect_elem.set('fill', '#CCCCCCFF')

            if bar == 'F':  # Full
                rect_elem.set('x', str(x_pos))
                rect_elem.set('y', str(start_y))
                rect_elem.set('width', str(bar_width))
                rect_elem.set('height', str(height_full))
            elif bar == 'A':  # Ascender
                rect_elem.set('x', str(x_pos))
                rect_elem.set('y', str(start_y))
                rect_elem.set('width', str(bar_width))
                rect_elem.set('height', str(height_asc))
            elif bar == 'D':  # Descender
                rect_elem.set('x', str(x_pos))
                rect_elem.set('y', str(start_y + height_full - height_desc))
                rect_elem.set('width', str(bar_width))
                rect_elem.set('height', str(height_desc))
            elif bar == 'T':  # Tracker
                rect_elem.set('x', str(x_pos))
                rect_elem.set('y', str(y_tracker))
                rect_elem.set('width', str(bar_width))
```

```
                rect_elem.set('height', str(tracker_height))

            root.append(rect_elem)
            x_pos += bar_width + space

        # Add label text below barcode if label_text is provided
        if label_text:
            if svg_namespace:
                text_elem = ET.Element('{' + svg_namespace + '}text')
            else:
                text_elem = ET.Element('text')
            text_elem.set('x', str(start_x))
            text_elem.set('y', str(start_y + height_full + 15))
            text_elem.set('font-family', 'Arial, sans-serif')
            text_elem.set('font-size', '12')
            text_elem.set('fill', '#000000')
            text_elem.text = label_text
            root.append(text_elem)

    # Update ALL existing text elements with label "auspost_label" (one per page)
    for auspost_label_text in auspost_label_texts:
        # Use label_text if provided, otherwise use barcode_data
        display_text = label_text if label_text is not None else barcode_data

        # Clear any existing text content from the text element itself
        auspost_label_text.text = None
        # Find the tspan element inside the text element and update it
        tspan_found = False
        for tspan in auspost_label_text.iter():
            tspan_tag = tspan.tag.split('}')[-1] if '}' in tspan.tag else tspan.tag
            if tspan_tag == 'tspan':
                # Update the text content in the tspan
                tspan.text = display_text
                tspan_found = True
                break
        # If no tspan found, create one or set text directly
        if not tspan_found:
            # Create a tspan element if it doesn't exist
            if svg_namespace:
                tspan_elem = ET.Element('{' + svg_namespace + '}tspan')
            else:
                tspan_elem = ET.Element('tspan')
            tspan_elem.text = display_text
            auspost_label_text.append(tspan_elem)

    # Save the modified SVG
    # Pretty print the XML
    xml_str = ET.tostring(root, encoding='unicode')
    dom = minidom.parseString(xml_str)
    pretty_xml = dom.toprettyxml(indent="  ")

    # Remove the XML declaration line added by minidom if the original didn't have
    with open(input_svg_path, 'r', encoding='utf-8') as f:
        original_content = f.read()
        has_xml_declaration = original_content.strip().startswith('<?xml')
```

```
    if not has_xml_declaration:
        # Remove the XML declaration
        lines = pretty_xml.split('\n')
        if lines[0].startswith('<?xml'):
            pretty_xml = '\n'.join(lines[1:])

    with open(output_svg_path, 'w', encoding='utf-8') as f:
        f.write(pretty_xml)

# Example usage:
# generate_4state_barcode_svg("HELLO123", "hello_barcode.svg")
# insert_barcode_into_svg("template.svg", "HELLO123", "output.svg")


def sign_pdf_with_barcodes(pdf_path, csv_path='v4_uuids.csv', output_path=None, out
    """
    Process a PDF document: overlay each page onto template SVG, add unique barcode

    Args:
        pdf_path: Path to the input PDF file
        csv_path: Path to the v4_uuids.csv file
        output_path: Path for the final merged PDF (defaults to input name with '_s
        output_dir: Directory for temporary files (defaults to same as PDF)
        template_svg_path: Path to the calibration template SVG (default: calibrati

    Returns:
        tuple: (output_path, list of (page_num, uuid, temp_pdf_path) tuples)
    """
    if not HAS_PYMUPDF:
        raise ImportError("PyMuPDF (fitz) is required. Install with: pip install Py

    # Check that at least one of the page-specific templates exists
    if not os.path.exists('calibration_page-coloured-0.svg') and not os.path.exists
        # Fallback: check if default template exists
        if not os.path.exists(template_svg_path):
            raise FileNotFoundError(f"Template SVGs not found: calibration_page-col

    if output_dir is None:
        output_dir = os.path.dirname(pdf_path) or '.'

    if output_path is None:
        base_name = os.path.splitext(os.path.basename(pdf_path))[0]
        output_path = os.path.join(output_dir, f"{base_name}_signed.pdf")

    # Read the CSV file
    rows = []
    with open(csv_path, 'r', encoding='utf-8') as f:
        reader = csv.DictReader(f)
        rows = list(reader)

    # Open the PDF
    doc = fitz.open(pdf_path)
    num_pages = len(doc)

    signed_pages = []
    temp_files = []
```

```
try:
    # Process each page by overlaying onto template
    for page_num in range(num_pages):
        # Find next available UUID
        selected_uuid = None
        selected_index = None
        for i, row in enumerate(rows):
            if not row.get('entity', '').strip() and not row.get('state', '').s
                selected_uuid = row['uuid']
                selected_index = i
                break

        if selected_uuid is None:
            print(f"Warning: No more UUIDs available for page {page_num + 1}")
            break

        # Get the PDF page (preserve vector content)
        page = doc[page_num]
        page_rect = page.rect

        # Select the correct template file based on even/odd page numbers
        # Odd pages (1, 3, 5...) → calibration_page-coloured-1.svg (which has "
        # Even pages (0, 2, 4...) → calibration_page-coloured-0.svg (which has
        if page_num % 2 == 1:  # Odd page (1-indexed: 1, 3, 5...)
            page_template_path = 'calibration_page-coloured-1.svg'
            template_page_label = "page 1"
        else:  # Even page (0-indexed: 0, 2, 4...)
            page_template_path = 'calibration_page-coloured-0.svg'
            template_page_label = "page 0"

        # Check if the page-specific template exists, fallback to default templ
        if not os.path.exists(page_template_path):
            if os.path.exists(template_svg_path):
                page_template_path = template_svg_path
            else:
                raise FileNotFoundError(f"Template SVG not found: {page_templat

        # Read template SVG to get auspost rect position and render as backgrou
        tree = ET.parse(page_template_path)
        root = tree.getroot()

        # Get SVG dimensions - try viewBox first (most reliable)
        if 'viewBox' in root.attrib:
            viewbox_parts = root.attrib['viewBox'].split()
            if len(viewbox_parts) >= 4:
                svg_width = float(viewbox_parts[2])
                svg_height = float(viewbox_parts[3])
            else:
                # Fallback to width/height attributes
                width_attr = root.get('width', '816')
                height_attr = root.get('height', '1056')
                # Handle units like '8.5in'
                if 'in' in str(width_attr):
                    svg_width = float(str(width_attr).replace('in', '').strip()
                else:
```

```
                    svg_width = float(width_attr)
                if 'in' in str(height_attr):
                    svg_height = float(str(height_attr).replace('in', '').strip
                else:
                    svg_height = float(height_attr)
            else:
                # Parse width/height attributes with unit handling
                width_attr = root.get('width', '816')
                height_attr = root.get('height', '1056')
                # Handle units like '8.5in'
                if 'in' in str(width_attr):
                    svg_width = float(str(width_attr).replace('in', '').strip()) *
                else:
                    svg_width = float(width_attr)
                if 'in' in str(height_attr):
                    svg_height = float(str(height_attr).replace('in', '').strip())
                else:
                    svg_height = float(height_attr)

            # Find auspost rect and auspost_label text in the appropriate template
            auspost_rect = None
            auspost_label_text = None
            target_page_layer = None

            # First, find the target page layer
            for elem in root.iter():
                tag = elem.tag.split('}')[-1] if '}' in elem.tag else elem.tag
                if tag == 'g':
                    elem_id = elem.get('id', '')
                    inkscape_label = None
                    for attr_name, attr_value in elem.attrib.items():
                        if attr_name.endswith('label') and attr_value == template_p
                            inkscape_label = attr_value
                            break
                    if elem_id in ['layer2', 'layer3'] or inkscape_label == templat
                        target_page_layer = elem
                        break

            # Search for auspost elements - check both the page layer and meta laye
            def find_auspost_elements(elem):
                nonlocal auspost_rect, auspost_label_text
                tag = elem.tag.split('}')[-1] if '}' in elem.tag else elem.tag
                if tag == 'rect':
                    elem_id = elem.get('id', '')
                    inkscape_label = None
                    for attr_name, attr_value in elem.attrib.items():
                        if attr_name.endswith('label') and attr_value == 'auspost':
                            inkscape_label = attr_value
                            break
                    if elem_id == 'auspost' or inkscape_label == 'auspost':
                        auspost_rect = elem
                elif tag == 'text':
                    inkscape_label = None
                    for attr_name, attr_value in elem.attrib.items():
                        if attr_name.endswith('label') and attr_value == 'auspost_l
                            inkscape_label = attr_value
```

```
                    break
            if inkscape_label == 'auspost_label':
                auspost_label_text = elem
        for child in elem:
            find_auspost_elements(child)

    # Search in the target page layer if found
    if target_page_layer is not None:
        find_auspost_elements(target_page_layer)

    # Also search in meta layers (layer1, layer4) which contain auspost ele
    for elem in root.iter():
        tag = elem.tag.split('}')[-1] if '}' in elem.tag else elem.tag
        if tag == 'g':
            elem_id = elem.get('id', '')
            inkscape_label = None
            for attr_name, attr_value in elem.attrib.items():
                if attr_name.endswith('label'):
                    inkscape_label = attr_value
                    break
            # Check for meta layers (page 0 meta, page 1 meta)
            if elem_id in ['layer1', 'layer4'] or (inkscape_label and 'meta
                # Check if this meta layer matches our page
                if (page_num % 2 == 0 and ('page 0' in str(inkscape_label)
                   (page_num % 2 == 1 and ('page 1' in str(inkscape_label)
                    find_auspost_elements(elem)

    # Fallback: if still not found, search entire document
    if auspost_rect is None:
        find_auspost_elements(root)

    if auspost_rect is None:
        raise ValueError("Could not find 'auspost' rect in template SVG")

    # Get auspost rect position and dimensions
    auspost_x = float(auspost_rect.get('x', 0))
    auspost_y = float(auspost_rect.get('y', 0))
    auspost_width = float(auspost_rect.get('width', 200))
    auspost_height = float(auspost_rect.get('height', 50))

    # Check if the parent layer has a transform that affects coordinates
    # For calibration_page-coloured-1.svg, layer3 and layer4 have transform
    # We need to find the transform by checking parent layers
    import re
    # Find the parent layer (g element) that contains the auspost_rect
    parent_layer = None
    for elem in root.iter():
        if elem == auspost_rect:
            continue
        # Check if auspost_rect is a child of this element
        for child in elem:
            if child == auspost_rect:
                tag = elem.tag.split('}')[-1] if '}' in elem.tag else elem.
                if tag == 'g':
                    parent_layer = elem
                    break
```

```
            if parent_layer is not None:
                break

        # Check for transform in parent layer and apply it
        label_transform_x = 0
        label_transform_y = 0
        if parent_layer is not None:
            transform_attr = parent_layer.get('transform', '')
            if transform_attr and 'translate' in transform_attr:
                # Extract translate values (e.g., "translate(-880)" or "transla
                match = re.search(r'translate\(([^)]+)\)', transform_attr)
                if match:
                    translate_values = match.group(1).split(',')
                    translate_x = float(translate_values[0].strip())
                    translate_y = float(translate_values[1].strip()) if len(tra
                    auspost_x += translate_x
                    auspost_y += translate_y
                    # Store transform for label text adjustment
                    label_transform_x = translate_x
                    label_transform_y = translate_y

        # Get auspost_label text position if it exists
        label_x = None
        label_y = None
        if auspost_label_text is not None:
            label_x = float(auspost_label_text.get('x', auspost_x))
            label_y = float(auspost_label_text.get('y', auspost_y + auspost_hei
            # Check for tspan inside text element
            for tspan in auspost_label_text.iter():
                tspan_tag = tspan.tag.split('}')[-1] if '}' in tspan.tag else t
                if tspan_tag == 'tspan':
                    tspan_x = tspan.get('x')
                    tspan_y = tspan.get('y')
                    if tspan_x is not None:
                        label_x = float(tspan_x)
                    if tspan_y is not None:
                        label_y = float(tspan_y)
                    break
            # Apply the same transform to label coordinates
            label_x += label_transform_x
            label_y += label_transform_y

        # Calculate scaling factor from SVG to PDF page
        scale_x = page_rect.width / svg_width
        scale_y = page_rect.height / svg_height

        # Scale auspost position to PDF coordinates
        pdf_auspost_x = auspost_x * scale_x
        pdf_auspost_y = auspost_y * scale_y
        pdf_auspost_width = auspost_width * scale_x
        pdf_auspost_height = auspost_height * scale_y

        # Scale label position to PDF coordinates
        pdf_label_x = label_x * scale_x if label_x is not None else None
        pdf_label_y = label_y * scale_y if label_y is not None else None
```

```
            # Hardcoded default font style as backup (used for SVG background)
            # Change DEFAULT_FONT_FAMILY to your desired font
            DEFAULT_FONT_FAMILY = 'Space Mono'  # Font name for SVG
            DEFAULT_FONT_SIZE = '13.3333px'
            DEFAULT_FILL_COLOR = '#cccccc'

            # Update the label text in the SVG template (so it's part of the backgr
            if auspost_label_text is not None:
                import re
                # Set font on the parent text element
                text_style = auspost_label_text.get('style', '')
                if text_style:
                    # Ensure font-family is in text element style
                    if 'font-family' not in text_style:
                        if text_style and not text_style.endswith(';'):
                            text_style += ';'
                        text_style += f'font-family:{DEFAULT_FONT_FAMILY}'
                    else:
                        # Override font-family in text element
                        text_style = re.sub(r'font-family:[^;]+', f'font-family:{DE
                    auspost_label_text.set('style', text_style)
                else:
                    # Create style for text element
                    text_style_parts = [
                        f'font-size:{DEFAULT_FONT_SIZE}',
                        f'font-family:{DEFAULT_FONT_FAMILY}',
                        f'fill:{DEFAULT_FILL_COLOR}',
                        f'fill-opacity:1'
                    ]
                    auspost_label_text.set('style', ';'.join(text_style_parts))

                # Also set font-family as a direct attribute
                auspost_label_text.set('font-family', DEFAULT_FONT_FAMILY)

                # Find existing tspan and update its text content
                tspan_found = False
                for tspan in auspost_label_text.iter():
                    tag_name = tspan.tag.split('}')[-1] if '}' in tspan.tag else ts
                    if tag_name == 'tspan':
                        # Update text content
                        tspan.text = selected_uuid
                        tspan.tail = None

                        # Ensure style attribute has font-family
                        original_style = tspan.get('style', '')
                        if original_style:
                            original_style = re.sub(r'font-family:[^;]+', f'font-fa
                            if 'font-family' not in original_style:
                                if original_style and not original_style.endswith('
                                    original_style += ';'
                                original_style += f'font-family:{DEFAULT_FONT_FAMIL
                            tspan.set('style', original_style)
                        else:
                            # Create style with defaults
                            style_parts = [
                                f'font-size:{DEFAULT_FONT_SIZE}',
```

```
                        f'font-family:{DEFAULT_FONT_FAMILY}',
                        f'fill:{DEFAULT_FILL_COLOR}',
                        f'fill-opacity:1'
                    ]
                    tspan.set('style', ';'.join(style_parts))
                tspan_found = True
                break

        # If no tspan found, create one
        if not tspan_found:
            svg_namespace = None
            if root.tag.startswith('{'):
                svg_namespace = root.tag.split('}')[0].strip('{')
            else:
                svg_namespace = root.attrib.get('xmlns', 'http://www.w3.org

            if svg_namespace:
                tspan_elem = ET.Element('{' + svg_namespace + '}tspan')
            else:
                tspan_elem = ET.Element('tspan')

            tspan_x = auspost_label_text.get('x')
            tspan_y = auspost_label_text.get('y')
            if tspan_x:
                tspan_elem.set('x', tspan_x)
            if tspan_y:
                tspan_elem.set('y', tspan_y)

            tspan_elem.text = selected_uuid
            auspost_label_text.append(tspan_elem)

    # Render template as background image (underlay)
    # Save template to temp file for rendering
    temp_template_svg = os.path.join(output_dir, f"temp_template_{page_num}
    xml_str = ET.tostring(root, encoding='unicode')
    dom = minidom.parseString(xml_str)
    pretty_xml = dom.toprettyxml(indent="  ")
    with open(page_template_path, 'r', encoding='utf-8') as f:
        original_content = f.read()
        has_xml_declaration = original_content.strip().startswith('<?xml')
    if not has_xml_declaration:
        lines = pretty_xml.split('\n')
        if lines[0].startswith('<?xml'):
            pretty_xml = '\n'.join(lines[1:])
    with open(temp_template_svg, 'w', encoding='utf-8') as f:
        f.write(pretty_xml)
    temp_files.append(temp_template_svg)

    # Render template SVG to pixmap for background
    zoom = 2.0
    mat = fitz.Matrix(zoom, zoom)
    template_doc = fitz.open(temp_template_svg)
    template_page = template_doc[0]
    template_pix = template_page.get_pixmap(matrix=mat)
    template_doc.close()
```

```
            # Create new PDF page with template as background
            temp_pdf = os.path.join(output_dir, f"temp_page_{page_num}.pdf")
            new_doc = fitz.open()
            new_page = new_doc.new_page(width=page_rect.width, height=page_rect.hei

            # Insert template as background image (underlay) - this includes the la
            new_page.insert_image(fitz.Rect(0, 0, page_rect.width, page_rect.height

            # Encode the barcode BEFORE overlaying PDF (so we have the dimensions)
            uuid_for_barcode = selected_uuid.replace('-', '')
            bars = encode_4state_barcode(uuid_for_barcode)
            num_bars = len(bars)

            # Calculate barcode dimensions to fit in auspost area
            # Make bars thinner with more spacing for better legibility
            available_width = pdf_auspost_width * 0.95
            # Make bars thinner by increasing the divisor
            bar_width = (available_width * 0.85) / (num_bars * 3) if num_bars > 0 e
            # Ensure minimum bar width for legibility
            if bar_width < 1.5:
                bar_width = 1.5
            space = bar_width * 1.0  # More space between bars
            total_barcode_width = num_bars * bar_width + (num_bars - 1) * space

            height_full = pdf_auspost_height * 0.9
            height_asc = height_full * 0.8
            height_desc = height_full * 0.8
            tracker_height = height_full * 0.2

            # Center barcode in auspost area
            barcode_start_x = pdf_auspost_x + (pdf_auspost_width - total_barcode_wi
            barcode_start_y = pdf_auspost_y + (pdf_auspost_height - height_full) /
            y_tracker = barcode_start_y + (height_full - tracker_height) / 2

            # Draw barcode bars BEFORE overlaying PDF (so it's in the background)
            # Use black color for maximum legibility
            barcode_color = (0.6, 0.6, 0.6)  # Dark gray (RGB values must be 0.0-1.
            x_pos = barcode_start_x
            for bar in bars:
                if bar == 'F':  # Full
                    rect = fitz.Rect(x_pos, barcode_start_y, x_pos + bar_width, bar
                    new_page.draw_rect(rect, color=barcode_color, fill=barcode_colo
                elif bar == 'A':  # Ascender
                    rect = fitz.Rect(x_pos, barcode_start_y, x_pos + bar_width, bar
                    new_page.draw_rect(rect, color=barcode_color, fill=barcode_colo
                elif bar == 'D':  # Descender
                    rect = fitz.Rect(x_pos, barcode_start_y + height_full - height_
                    new_page.draw_rect(rect, color=barcode_color, fill=barcode_colo
                elif bar == 'T':  # Tracker
                    rect = fitz.Rect(x_pos, y_tracker, x_pos + bar_width, y_tracker
                    new_page.draw_rect(rect, color=barcode_color, fill=barcode_colo
                x_pos += bar_width + space

            # Overlay original PDF page content on top (preserving vector)
            # Use show_pdf_page to insert the original page on top of background
            new_page.show_pdf_page(fitz.Rect(0, 0, page_rect.width, page_rect.heigh
```

```
            # Save the new PDF page
            new_doc.save(temp_pdf)
            new_doc.close()
            temp_files.append(temp_pdf)

            # Update CSV
            rows[selected_index]['entity'] = f"{os.path.basename(output_path)}_page
            rows[selected_index]['state'] = 'active'

            signed_pages.append((page_num + 1, selected_uuid, temp_pdf))

        # Merge all PDF pages back together
        merged_doc = fitz.open()
        for page_num, uuid, temp_pdf_path in signed_pages:
            page_doc = fitz.open(temp_pdf_path)
            merged_doc.insert_pdf(page_doc)
            page_doc.close()

        # Save merged PDF
        merged_doc.save(output_path)
        merged_doc.close()

        # Write back to CSV
        with open(csv_path, 'w', encoding='utf-8', newline='') as f:
            fieldnames = ['uuid', 'entity', 'state']
            writer = csv.DictWriter(f, fieldnames=fieldnames)
            writer.writeheader()
            writer.writerows(rows)

    finally:
        # Clean up temporary files
        for temp_file in temp_files:
            try:
                if os.path.exists(temp_file):
                    os.remove(temp_file)
            except Exception as e:
                print(f"Warning: Could not remove temp file {temp_file}: {e}")

    doc.close()

    return (output_path, signed_pages)


def sign_with_next_uuid(csv_path, input_svg_path, output_filename=None, output_dir=
    """
    Sign an SVG with the next available UUID from the CSV file.

    Args:
        csv_path: Path to the v4_uuids.csv file
        input_svg_path: Path to the input SVG file
        output_filename: Custom output filename (without extension). If None, uses
        output_dir: Directory to save output (defaults to same as input_svg_path)

    Returns:
        tuple: (uuid, output_path) or None if no available UUID found
```

```
    """
    # Read the CSV file
    rows = []
    with open(csv_path, 'r', encoding='utf-8') as f:
        reader = csv.DictReader(f)
        rows = list(reader)

    # Find the next available UUID (empty entity and state)
    selected_uuid = None
    selected_index = None
    for i, row in enumerate(rows):
        if not row.get('entity', '').strip() and not row.get('state', '').strip():
            selected_uuid = row['uuid']
            selected_index = i
            break

    if selected_uuid is None:
        return None

    # Remove hyphens from UUID for barcode encoding
    uuid_for_barcode = selected_uuid.replace('-', '')

    # Determine output path
    if output_dir is None:
        output_dir = os.path.dirname(input_svg_path) or '.'

    if output_filename is None:
        output_filename = selected_uuid.replace('-', '_')

    output_path = os.path.join(output_dir, f"{output_filename}.svg")

    # Generate the barcode (use UUID with hyphens for label, without hyphens for en
    insert_barcode_into_svg(input_svg_path, uuid_for_barcode, output_path, label_te

    # Update the CSV (store just the filename, not the full path)
    output_filename_with_ext = f"{output_filename}.svg"
    rows[selected_index]['entity'] = output_filename_with_ext
    rows[selected_index]['state'] = 'active'

    # Write back to CSV
    with open(csv_path, 'w', encoding='utf-8', newline='') as f:
        fieldnames = ['uuid', 'entity', 'state']
        writer = csv.DictWriter(f, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(rows)

    return (selected_uuid, output_path)


def sign_with_uuid(uuid, input_svg_path, output_filename=None, output_dir=None):
    """
    Sign an SVG with a specific UUID.

    Args:
        uuid: UUID string to use for signing
        input_svg_path: Path to the input SVG file
```

```
        output_filename: Custom output filename (without extension). If None, uses
        output_dir: Directory to save output (defaults to same as input_svg_path)

    Returns:
        tuple: (uuid, output_path)
    """
    # Remove hyphens from UUID for barcode encoding
    uuid_for_barcode = uuid.replace('-', '')

    # Determine output path
    if output_dir is None:
        output_dir = os.path.dirname(input_svg_path) or '.'

    if output_filename is None:
        output_filename = uuid.replace('-', '_')

    output_path = os.path.join(output_dir, f"{output_filename}.svg")

    # Generate the barcode (use UUID with hyphens for label, without hyphens for en
    insert_barcode_into_svg(input_svg_path, uuid_for_barcode, output_path, label_te

    return (uuid, output_path)


def main():
    """
    Command-line interface for signing documents with UUIDs.
    """
    parser = argparse.ArgumentParser(
        description='Sign an SVG document with a UUID barcode',
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog="""
Examples:
  # Use next available UUID from CSV
  python homework_signer.py

  # Use specific UUID
  python homework_signer.py --uuid "b09b61df-5c2e-442f-a6e8-0c8cb600642f"

  # Custom output filename
  python homework_signer.py --output "my_document"

  # Custom CSV file
  python homework_signer.py --csv "custom_uuids.csv"

  # Process PDF: extract pages, add barcode to each, merge back
  python homework_signer.py --pdf "document.pdf" --pdf-output "document_signed.pdf"
        """
    )

    parser.add_argument(
        '--uuid',
        type=str,
        help='Specific UUID to use for signing (if not provided, uses next availabl
    )
```

```
    parser.add_argument(
        '--csv',
        type=str,
        default='v4_uuids.csv',
        help='Path to CSV file with UUIDs (default: v4_uuids.csv)'
    )

    parser.add_argument(
        '--input',
        type=str,
        default='calibration_page-coloured.svg',
        help='Path to input SVG template (default: calibration_page-coloured.svg)'
    )

    parser.add_argument(
        '--output',
        type=str,
        help='Output filename without extension (default: uses UUID or next availab
    )

    parser.add_argument(
        '--output-dir',
        type=str,
        help='Directory to save output file (default: same as input file directory)
    )

    parser.add_argument(
        '--pdf',
        type=str,
        help='Process a PDF file: extract pages, add unique barcode to each, and me
    )

    parser.add_argument(
        '--pdf-output',
        type=str,
        help='Output path for signed PDF (default: input name with _signed suffix)'
    )

    args = parser.parse_args()

    try:
        if args.pdf:
            # Process PDF file
            if not os.path.exists(args.pdf):
                print(f"✗ Error: PDF file not found: {args.pdf}")
                sys.exit(1)

            if not os.path.exists(args.csv):
                print(f"✗ Error: CSV file not found: {args.csv}")
                sys.exit(1)

            output_path, signed_pages = sign_pdf_with_barcodes(
                pdf_path=args.pdf,
                csv_path=args.csv,
                output_path=args.pdf_output,
                output_dir=args.output_dir,
```

```
                template_svg_path=args.input
            )

            print(f"✓ Processed {len(signed_pages)} pages:")
            for page_num, uuid, _ in signed_pages:
                print(f"  Page {page_num}: UUID {uuid}")
            print(f"✓ Merged PDF saved to: {output_path}")
            print(f"✓ CSV updated with {len(signed_pages)} UUIDs")

        elif args.uuid:
            # Use specified UUID
            result = sign_with_uuid(
                uuid=args.uuid,
                input_svg_path=args.input,
                output_filename=args.output,
                output_dir=args.output_dir
            )
            uuid, output_path = result
            print(f"✓ Document signed with UUID: {uuid}")
            print(f"✓ Output saved to: {output_path}")
        else:
            # Use next available UUID from CSV
            if not os.path.exists(args.csv):
                print(f"✗ Error: CSV file not found: {args.csv}")
                sys.exit(1)

            result = sign_with_next_uuid(
                csv_path=args.csv,
                input_svg_path=args.input,
                output_filename=args.output,
                output_dir=args.output_dir
            )

            if result:
                uuid, output_path = result
                print(f"✓ Document signed with next available UUID: {uuid}")
                print(f"✓ Output saved to: {output_path}")
                print(f"✓ CSV updated: entity={os.path.basename(output_path)}, stat
            else:
                print("✗ Error: No available UUIDs found in CSV file")
                sys.exit(1)

    except Exception as e:
        import traceback
        print(f"✗ Error: {e}")
        if str(e) == "":
            traceback.print_exc()
        sys.exit(1)


if __name__ == '__main__':
    main()
```

In [ ]:
```
from homework_signer import sign_with_next_uuid

# Sign a document with the next available UUID
```

```python
# You can customize the output filename
result = sign_with_next_uuid(
    csv_path='v4_uuids.csv',
    input_svg_path='calibration_page-coloured.svg',
    output_filename='document_001'  # Customize this name
)

if result:
    uuid, output_path = result
    print(f"✓ Document signed successfully!")
    print(f"  UUID: {uuid}")
    print(f"  Output file: {output_path}")
    print(f"  CSV updated: entity={output_path}, state=active")
else:
    print("✗ No available UUID found in the CSV file")
```