

```

#define SKETCH_NAME F("camera-sd-5282a")
// Motivation: take pics at intervals accounting for picture elapsed time
// Features:
// Done: Verify function
// Done: button mode adjust loop delay to account for function elapsed time
// Done: Automatic mode loop delay

#include <Wire.h>
#include <ArduCAM.h>
#include <SPI.h>
#include <SD.h>
#include "memorysaver.h"

#define OV2640_MINI_ZMP
#define CS_CAM 10
#define CS_SD 9

#define BUTTON_PIN 7 // ☒ Added: button on digital pin 7
#define DIP0 4 // reversed so 100 means left switch up
#define DIP1 3 // see switch dipValue cases below
#define DIP2 2

#define JPEG_SIZE OV2640_1600x1200

ArduCAM myCAM(OV2640, CS_CAM);

enum ModeType { MODE_AUTO, MODE_BUTTON };
ModeType mode;
unsigned long intervalMs = 0;

const char* getResolutionName(uint8_t res) {
  switch (res) {
    case 0: return "160x120";
    case 1: return "176x144";
    case 2: return "320x240";
    case 3: return "352x288";
    case 4: return "640x480";
    case 5: return "800x600";
    case 6: return "1024x768";
    case 7: return "1280x1024";
    case 8: return "1600x1200";
    default: return "Unknown";
  }
}

String toBinaryString(unsigned int value, byte places) {
  String binary = "";
  for (int i = places - 1; i >= 0; i--) {
    binary += bitRead(value, i);
  }
  return binary;
}

bool isBitSet(uint8_t value, uint8_t bitPosition) {
  return (value & (1 << bitPosition)) != 0;
}

// Read DIP switch value
int readDipSwitch() {
  pinMode(DIP0, INPUT_PULLUP);
  pinMode(DIP1, INPUT_PULLUP);
  pinMode(DIP2, INPUT_PULLUP);

  int dipValue = ((digitalRead(DIP2) == LOW) ? 1 : 0) << 2 |
    ((digitalRead(DIP1) == LOW) ? 1 : 0) << 1 |
    ((digitalRead(DIP0) == LOW) ? 1 : 0);

  return dipValue;
}

// Set mode and display based on DIP value
void displayDipValue(int dipValue) {
  switch (dipValue) {
    case 0b000: mode = MODE_AUTO; intervalMs = 15000; break; // 15 secs
    case 0b010: mode = MODE_AUTO; intervalMs = 15000; break; // 15 secs
    case 0b001: mode = MODE_AUTO; intervalMs = 60000; break; // 1 min
    case 0b011: mode = MODE_AUTO; intervalMs = 900000; break; // 15 mins

    case 0b111: mode = MODE_BUTTON; intervalMs = 15000; break; // 15 secs
    case 0b100: mode = MODE_BUTTON; intervalMs = 15000; break; // 15 secs
    case 0b110: mode = MODE_BUTTON; intervalMs = 60000; break; // 1 min
    case 0b101: mode = MODE_BUTTON; intervalMs = 900000; break; // 15 mins

    default: mode = MODE_BUTTON; intervalMs = 15000; break; // 15 seconds
  }
}

// intervalMs = 0; // no delay to see natural time for loop
// intervalMs = 2000; // Override quick w delay
// checking the pin time since w low numbers it's running at 4 secs per picture default since

printElapsedTime(millis() / 1000, intervalMs);
Serial.print(F("Mode: "));
Serial.print(toBinaryString(dipValue, 3));
Serial.print(F(", "));
Serial.print(mode == MODE_AUTO ? "AUTO" : "BUTTON");
Serial.print(F(", interval: "));
Serial.print(intervalMs);
Serial.println(F(" ms"));
}

void setup() {
  uint8_t vid, pid;
  int dipValue;

  Serial.begin(115200);
  delay(1000);
  Wire.begin();

  pinMode(CS_CAM, OUTPUT);
  pinMode(CS_SD, OUTPUT);
  digitalWrite(CS_CAM, HIGH);
  digitalWrite(CS_SD, HIGH);

  pinMode(BUTTON_PIN, INPUT_PULLUP); // ☒ Added: setup button pin

  dipValue = readDipSwitch(); // read early so can use intervalMs in printElapsedTime

  SPI.begin();
  Serial.println(F(""));
  printElapsedTime(millis() / 1000, intervalMs);
  printElapsedTime(millis() / 1000, intervalMs);
  Serial.print(F("Running sketch: "));
  Serial.println(String(SKETCH_NAME));

  if (!SD.begin(CS_SD)) {
    Serial.println(F("SD Card init failed!"));
    while (1);
  }

  printElapsedTime(millis() / 1000, intervalMs);
  Serial.println(F("SD Card initialized"));

  printElapsedTime(millis() / 1000, intervalMs);
  Serial.print(F("Using set_JPEG_size: "));
  Serial.println(String(getResolutionName(JPEG_SIZE)));

  myCAM.write_reg(ARDUCHIP_MODE, 0x00); // switch to MCU mode
  myCAM.initCAM();
  myCAM.set_format(JPEG);
  myCAM.initCAM();
  myCAM.OV2640_set_JPEG_size(JPEG_SIZE); // Set resolution

  // dipValue = readDipSwitch();

  displayDipValue(dipValue);

  // 1xx button mode; 0xx automatic
  if (isBitSet(dipValue, 2) ) {
    printElapsedTime(millis() / 1000, intervalMs);
    Serial.print(F("Button state: "));
    Serial.println(digitalRead(BUTTON_PIN));

    printElapsedTime(millis() / 1000, intervalMs);
    Serial.println(F("Ready. Press button to take pictures."));
    Serial.println(F(""));
  } else {
    Serial.println(F("Automatic. Taking pictures."));
  }
}

void loop() {
  if (isBitSet(readDipSwitch(), 2) ) {
    buttonMode();
  } else {
    automaticMode();
  }
}

void automaticMode() {
  static unsigned long lastPictureTime = 0;
  const unsigned long interval = intervalMs;

  if ((millis() - lastPictureTime >= interval)) {
    unsigned long t0 = millis();
    takePicture();
    unsigned long elapsed = millis() - t0;
    lastPictureTime = millis() - elapsed; // Adjust for picture time
  }
}

void buttonMode() {
  static bool lastStableState = HIGH;
  static bool lastReadState = HIGH;
  static unsigned long lastDebounceTime = 0;
  static bool takingPictures = false;
  static unsigned long lastPictureTime = 0;

  const unsigned long debounceDelay = 50;
  const unsigned long interval = intervalMs; // 5 seconds

  bool currentRead = digitalRead(BUTTON_PIN);

  // Debounce logic
  if (currentRead != lastReadState) {
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (currentRead != lastStableState) {
      lastStableState = currentRead;
      if (currentRead == LOW) {
        // Toggle picture taking mode
        takingPictures = !takingPictures;
        printElapsedTime(millis() / 1000, intervalMs);
        Serial.print(F("Picture taking mode: "));
        Serial.println(takingPictures ? F("ON") : F("OFF"));
        takingPictures && (printElapsedTime(F("Camera loop starting"), millis() / 1000, intervalMs), true);

        Serial.println(F(""));
        // Reset timer when toggled on
        if (takingPictures) {
          lastPictureTime = millis();
        }
      }
    }
  }

  if (takingPictures && (millis() - lastPictureTime >= interval)) {
    unsigned long t0 = millis();
    takePicture();
    unsigned long elapsed = millis() - t0;
    lastPictureTime = millis() - elapsed; // Adjust for picture time
  }

  lastReadState = currentRead;
}

// Base version (no title)
void printElapsedTime(unsigned long seconds, unsigned long interval) {
  char buf[5]; // 4 digits + null terminator
  sprintf(buf, "%04u", seconds); // 'u' for unsigned
  // print format elapsed secs: interval ms e.g. [0001:2000]
  Serial.print(F(""));
  Serial.print(buf);
  Serial.print(F(": "));
  Serial.print(interval);
  Serial.print(F(": "));
}

// Overload for F() titles
void printElapsedTime(const __FlashStringHelper* title, unsigned long seconds, unsigned long interval) {
  printElapsedTime(seconds, interval); // reuse base version
  Serial.println(title); // prints from flash
}

void takePicture() {
  unsigned long seconds = millis() / 1000;

  printElapsedTime(F("takePicture"), seconds, intervalMs);

  myCAM.clear_fifo_flag();
  myCAM.start_capture();

  while (!myCAM.get_bit(ARDUCHIP_TRIG, CAP_DONE_MASK));
  printElapsedTime(F("Capture done"), seconds, intervalMs);

  saveImageToSD();
}

void saveImageToSD() {
  char prefix = 'B'; // Set this manually to any letter A-Z
  char filename[15]; // Enough space for "/A0000000.jpg" + null terminator

  // Initialize filename with prefix and zeroed number
  sprintf(filename, "%c00000000.jpg", prefix);

  for (unsigned long num = 0; num <= 9999999; num++) {
    sprintf(&filename[2], "%07lu.jpg", num); // Update numeric part and extension
    if (!SD.exists(filename)) {
      printElapsedTime(millis() / 1000, intervalMs);
      Serial.print("Saving to: "); // can
      Serial.println(filename);
      // Serial.print("-"); // can
      break;
    }
  }

  File imgFile = SD.open(filename, FILE_WRITE);
  if (!imgFile) {
    Serial.println(F("Failed to open file"));
    return;
  }

  uint32_t length = myCAM.read_fifo_length();
  if (length >= 0x7FFFFF || length == 0) {
    Serial.println(F("Invalid image length"));
    imgFile.close();
    return;
  }

  myCAM.CS_LOW();

  bool foundFFD8 = false;
  bool lastWasFF = false;
  uint8_t b;

  const int BUF_SIZE = 256; // Original buffer size above sizes mad no difference
  uint8_t buf[BUF_SIZE];
  int buf_idx = 0;

  for (uint32_t i = 0; i < length; i++) {
    b = myCAM.read_fifo();

    if (!foundFFD8) {
      if (lastWasFF && b == 0xD8) {
        buf[buf_idx++] = 0xFF;
        buf[buf_idx++] = 0xD8;
        foundFFD8 = true;
        lastWasFF = false;
        continue;
      }
      lastWasFF = (b == 0xFF);
      continue;
    }

    buf[buf_idx++] = b;

    if (buf_idx >= BUF_SIZE) {
      imgFile.write(buf, buf_idx);
      buf_idx = 0;
    }

    if (lastWasFF && b == 0xD9) {
      if (buf_idx > 0) imgFile.write(buf, buf_idx);
      printElapsedTime(F("Found FFD9, done"), millis() / 1000, intervalMs);
      break;
    }

    lastWasFF = (b == 0xFF);
  }

  if (buf_idx > 0) {
    imgFile.write(buf, buf_idx);
  }

  myCAM.CS_HIGH();
  imgFile.close();

  printElapsedTime(millis() / 1000, intervalMs);
  Serial.println(F("☒ JPEG saved using ArduCam read_fifo()));
  Serial.println(F(""));
}

```